Structure-aware Fine-tuning of Sequence-to-sequence Transformers for Transition-based AMR Parsing

Jiawei Zhou[♠] **Tahira Naseem**[◇] **Ramón Fernandez Astudillo**[◇] **Young-Suk Lee**[◇]
Radu Florian[◇] **Salim Roukos**[◇]
[♠]Harvard University [◇]IBM Research
[♠]jzhou02@g.harvard.edu [◇]ramon.astudillo@ibm.com
[◇]{tnaseem, ysuklee, raduf, roukos}@us.ibm.com

Abstract

Predicting linearized Abstract Meaning Representation (AMR) graphs using pre-trained sequence-to-sequence Transformer models has recently led to large improvements on AMR parsing benchmarks. These parsers are simple and avoid explicit modeling of structure but lack desirable properties such as graph well-formedness guarantees or built-in graph-sentence alignments. In this work we explore the integration of general pre-trained sequence-to-sequence language models and a structure-aware transition-based approach. We depart from a pointer-based transition system and propose a simplified transition set, designed to better exploit pre-trained language models for structured fine-tuning. We also explore modeling the parser state within the pre-trained encoder-decoder architecture and different vocabulary strategies for the same purpose. We provide a detailed comparison with recent progress in AMR parsing and show that the proposed parser retains the desirable properties of previous transition-based approaches, while being simpler and reaching the new parsing state of the art for AMR 2.0, without the need for graph re-categorization.

1 Introduction

The task of Abstract Meaning Representation (AMR) parsing translates a natural sentence into a rooted directed acyclic graph capturing the semantics of the sentence, with nodes representing concepts and edges representing their relations (Banarescu et al., 2013). Recent works utilizing pre-trained encoder-decoder language models show great improvements in AMR parsing results (Xu et al., 2020; Bevilacqua et al., 2021). These approaches avoid explicit modeling of the graph structure. Instead, they directly predict the linearized AMR graph treated as free text. While the use of pre-trained Transformer encoders is widely extended in AMR parsing, the use of pre-trained Transformer decoders is recent and has shown to be very effective, maintaining current state-of-the-art results (Bevilacqua et al., 2021).

These approaches however lack certain desirable properties. There are no structural guarantees of graph well-formedness, i.e. the model may predict strings that can not be decoded into valid graphs, and post-processing is required. Furthermore, predicting AMR linearizations ignores the implicit alignments between graph nodes and words, which provide a strong inductive bias and are useful for downstream AMR applications (Mitra and Baral, 2016; Liu et al., 2018; Vlachos et al., 2018; Kapanipathi et al., 2021; Naseem et al., 2021).

On the other hand, transition-based AMR parsers (Wang et al., 2015; Ballesteros and Al-Onaizan, 2017a; Astudillo et al., 2020; Zhou et al., 2021) operate over the tokens of the input sentence, generating the graph incrementally. They implicitly model graph structural constraints through transitions and yield alignments by construction, thus guaranteeing graph well-formedness.[1] However, it remains unclear whether explicit modeling of structure is still beneficial for AMR parsing in the presence of powerful pre-trained language models and their strong free text generation abilities.

In this work, we integrate pre-trained sequence-to-sequence (seq-to-seq) language models with the transition-based approach for AMR parsing, and explore to what degree they are complementary. To fully utilize the generation power of the pre-trained language models, we propose a transition system with a small set of basic actions – a generalization of the action-pointer transition system of Zhou et al. (2021). We use BART (Lewis et al., 2019) as our pre-trained language model, since it has shown significant improvements in linearized AMR generation (Bevilacqua et al., 2021). Unlike previous approaches that directly fine-tune the model with

[1]With the only exception being disconnected graphs, which happen infrequently in practice.

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 6279–6290
November 7–11, 2021.

linearized graphs, we modify the model structure to work with our transition system, and encode parser states in BART's attention mechanism (Astudillo et al., 2020; Zhou et al., 2021). We also explore different vocabulary strategies for action generation. These changes convert the pre-trained BART to a transition-based parser where graph constraints and alignments are internalized.

We provide a detailed comparison with top-performing AMR parsers and perform ablation experiments showing that our proposed transition system and BART modifications are both necessary to achieve strong performance. Although BART has great language generation capacity, it still benefits from parser state encoding with hard attention, and can efficiently learn structural output. Our model establishes a new state of the art for AMR 2.0 while maintaining graph well-formedness guarantees and producing built-in alignments.

2 Intricacies of AMR Parsers

A frequent complaint about AMR parsers is that they involve combining many different techniques and hand-crafted rules, resulting in complex pipelines that are hard to analyze and generalize poorly. This situation has notably improved in the past few years but there are still two main sources of complexity present in almost all recent parsers: graph re-categorization and subgraph actions.

Graph re-categorization (Wang and Xue, 2017; Lyu and Titov, 2018; Zhang et al., 2019a) normalizes the graph prior to learning. This includes joining certain subgraphs such as entities, dates and other constructs into single nodes, removing special types of nodes like polarity and normalizing propbank names. An example of common normalizations is displayed in Figure 1. Training and decoding of models using this technique happens in this re-categorized space. Re-categorized graphs are expanded to normal valid AMR graphs in a post-processing stage. The type and number of subgraphs normalized vary across implementations, but most high performing approaches (Cai and Lam, 2020; Bevilacqua et al., 2021) utilize the re-categorization described in Appendix A.1 of Zhang et al. (2019a). This version requires of an external Named Entity Recognition (NER) system to anonymize named entities, both at train time and test time. It also makes use of look-up tables for nominalizations (e.g. *English* to *England*) and other hand-crafted rules. Graph re-categorization

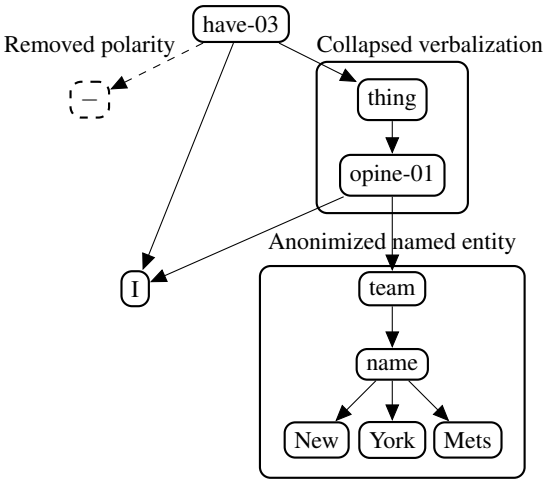

Figure 1: AMR graph of the sentence *I have no opinion on the New York Mets*. Examples of subgraphs for entity anonymization, collapsing of verbalized nouns and removal of the polarity node and edge.

has been criticised for its lack of generalization to new domains, such as biomedical domain or even the AMR 3.0 corpus (Bevilacqua et al., 2021). Recent top performing systems e.g. Cai and Lam (2020); Bevilacqua et al. (2021) also provide results without re-categorization, but this is shown to hurt performance notably on the AMR 2.0 corpus.

Subgraph actions (Ballesteros and Al-Onaizan, 2017b) are used in transition-based systems and play a role similar to re-categorization. Instead of normalizing and reverting, transition-based parsers apply a subgraph action that generates an entire subgraph at once. This subgraph action coincides with many of the subgraphs collapsed in re-categorization. Subgraph actions bring however fewer external dependencies, since the parser learns to segment and identify subgraphs during training. They still suffer however from data sparsity since some subgraphs appear very few times. As in re-categorization, subgraph actions also make use of lookup tables for nominalization and similar constructs that hinder generalization. Furthermore, they create the problem of unattachable nodes. This was addressed in Zhou et al. (2021) by ignoring subgraphs for a set of heuristically determined cases. Subgraph actions have been used in all transition-based AMR systems (Naseem et al., 2019; Astudillo et al., 2020; Zhou et al., 2021).

Aside from NER, past AMR parsers have other external dependencies such as POS taggers (Zhang et al., 2019a; Cai and Lam, 2020) and lemmatizers (Cai and Lam, 2020; Naseem et al., 2019; Astudillo et al., 2020).

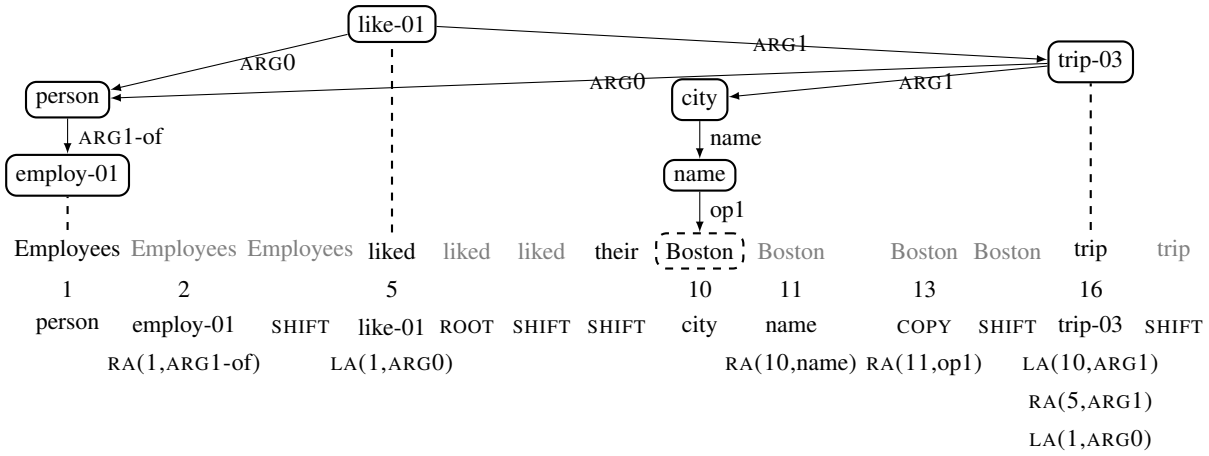

Figure 2: From top to bottom: graph (solid lines), sentence (source), addressable action positions and action sequence (target) for the sentence *Employees liked their Boston trip*, aligned (dotted lines) to its AMR graph. Arc-creating actions are displayed vertically due to space constraints. Words are repeated in grey to indicate the word under cursor for each action. The node *Boston* in dotted box is created by copying the token under cursor via COPY action at position 13. LA(1,ARG0) creates a left arc with label ARG0 from the top concept *like-01* to the concept *person* at position 1. For the concept *trip-03*, LA(1,ARG0) is a co-reference (re-entrancy) to the concept *person*.

3 A Simplified Transition System

In this section we propose a transition system for AMR parsing designed with two objectives: maximize the use of strong pre-trained decoders such as BART, and minimize the complexity and dependencies of the transition system compared to previous approaches. Similarly to Zhou et al. (2021), we scan the sentence from left to right and use a *token cursor* to point to a source token at each step. Parser actions can either shift the cursor one token forward or generate any number of nodes and edges while the cursor points to the same token. The proposed set of actions is as follows:

SHIFT moves token cursor one word to the right.

<string> creates node of name <string>.

COPY creates node where the node name is the token under the current cursor position.

LA(j,LBL) creates an arc with label LBL from the last generated node to the node generated at the j_{th} transition step.

RA(j,LBL) same as LA but with arc direction reversed.

ROOT declares the last predicted node as the root.

Unlike previous transition-based approaches, we do not use a reserved action, such as PRED (Zhou et al., 2021) or CONFIRM (Ballesteros and Al-Onaizan, 2017b), to predict nodes;[2] instead we directly use the node name <string> as the action symbol generating that node. This opens the possibility of utilizing BART's target side pre-trained vocabulary. We avoid using any copy actions that involve copying from lemmatizer outputs or lookup tables. Our COPY action is limited to copying the lower cased word. We also eliminate the use of SUBGRAPH (Zhou et al., 2021) or ENTITY (Ballesteros and Al-Onaizan, 2017b) actions producing multiple connected nodes simultaneously, as well as MERGE action creating spans of words. In previous approaches these actions were derived from alignments or hand-crafted. They thus did not cover all possible cases limiting the scalability of the approach. Finally, we discard the REDUCE action previously used to delete a source token. The effect can be achieved by simply using SHIFT without performing any other action. Figure 2 shows an example sentence with an action sequence and the corresponding graph. This can be compared with the handling of verbalization and named entities in Figure 1.

To train a parser with a transition system, we need an action sequence for each training sentence that will produce the gold graph when executed. This action sequence then serves as the target for seq-to-seq models. A simple rule-based *oracle* algorithm creates these ground-truth sequences given

[2] PRED wraps the node name inside the action as PRED(<string>), and CONFIRM calls a subroutine to predict the AMR node and find the right propbank sense.

a sentence, its AMR graph and node-to-word alignments. At each step, the oracle tries the following possibilities in the order of listing and performs the first one that applies:

1. Create next gold arc between last created node and previously created nodes;
2. Create next gold node aligned to token under cursor;
3. If not at sentence end, SHIFT source cursor;
4. Finish oracle.

If possible, nodes are generated by COPY and otherwise with <string> actions. Arcs are generated with LA and RA, connecting the nodes closer to the current node before the ones that are farther away. Note that the arcs are created by pointing to positions in the action history, where a graph node is represented by the action that creates it, following Zhou et al. (2021). Multiple nodes can be generated at a single source word before the cursor is moved by SHIFT. When multiple nodes are aligned to the same token, these nodes are generated in a predetermined topological order of graph traversal, interleaved by edge creation actions. ROOT is applied as soon as the root node is generated.

The above oracle circumvents the problem of unattachable nodes by avoiding the use of subgraph actions. This implies that the oracle will always produce a unique action sequence that fully recovers the gold graph as long as every node in the graph is aligned to some token. To guarantee that all nodes are aligned, we improve upon the alignment system from Naseem et al. (2019); Astudillo et al. (2020), which aligns a large majority, but not all AMR nodes.[3] In order to do this, we apply a heuristic based on graph proximity to maintain local correspondences between graph nodes and sentence words. If a node is not aligned, we copy the alignment from its first child node, if existing, and otherwise the alignment from its first parent node. For example, in Figure 2 if the node *person* was not provided with an alignment, our oracle would have assigned it to the aligned token of its child node *employ-01*. This is a recursive procedure – as long as there are some alignments to start with and the ground-truth graph is connected, all the nodes will get aligned.

[3]Roughly 1% of graphs contain about 1-2 unaligned nodes.

Our proposed transition system makes better use of BART pre-trained decoder compared to previous transition-based approaches (see Section 6) while greatly simplifying the transition set. It also naturally produces node-to-word alignments via source token cursor in the meantime.

4 Parsing Model

We build our model on top of the pre-trained seq-to-seq Transformer, BART (Lewis et al., 2019). We modify its architecture to incorporate a pointer network and internalize parser states induced by our transition system, and fine-tune for sentence-to-action generation.

4.1 Structure-aware Architecture

We adopt similar modifications on the Transformer architecture as in Zhou et al. (2021) since our transition system is based on the same action-pointer mechanism. The modifications do not introduce new modules or extra parameters, which naturally fit our need to adapt BART into a transition-based parser with internal graph well-formedness.

In particular, the target actions are factored into two parts: bare action symbols (containing labels when presented) and pointer values for edges. We use the BART standard output for the former, and a pointer network for the latter. As the pointing happens on the actions history, essentially a self-attention mechanism, we re-purpose one decoder self-attention head as the pointer network. It is supervised with additional cross entropy loss during fine-tuning and decoded for building graph edges at inference.

We encode the monotonic action-source alignments induced by the parser state with hard attention, i.e. by masking some decoder cross-attention heads to only focus on aligned words. Since BART processes source sentences with subwords, we apply an additional average pooling layer on top of its encoder to return states of original source words, used for the decoder layers for our hard attention. At last, as the possible valid actions are constrained with transition rules and states at every step, we restrict the decoder output space via hard masking of the BART final softmax layer. For simplicity, we do not incorporate the GNN-style (Li et al., 2019) step-wise decoder graph embedding technique in Zhou et al. (2021) as their gain was shown to be modest.

4.2 Action Generation

According to how we treat the target-side vocabulary for action generation, we propose two variations of the model. The first one is to use a completely separate vocabulary for target actions, where the decoder input side and output side use stand-alone embeddings for actions, separate from the pre-trained BART subword embeddings.[4] We denote this setup as our *sep-voc* model (abbreviated as StructBART-S).

However, this might not fully utilize the power of the pre-trained BART since it is an encoder-decoder model with a single vocabulary and all embeddings shared. Although our generation targets are action symbols, the node generating actions are closely related to natural words in their surface forms, which are what BART was pre-trained on. Therefore, we propose a second variation where we use a joint vocabulary for both the source tokens and target actions. Naively relying on the original BART subword vocabulary would end up splitting action symbols blindly, which is not desired as the structures such as alignments and edge pointers would be disrupted. Inspired by Bevilacqua et al. (2021), we add frequent node-creating actions to the vocabulary, in order to capture common AMR concepts intact, and split the remaining concepts with BART subword vocabulary. Non-node-creating actions such as SHIFT and COPY are added as-is to the BART vocabulary.

In this setup, a single node string can potentially be generated with multiple steps; we modify the arc transitions to always point to the beginning position of a node string for attachment. With joint vocabulary setup, the model could learn to generate unseen nodes with BART's subword vocabulary, eliminating potential out-of-vocabulary problems. We refer to this setup as our *joint-voc* model (abbreviated as StructBART-J).

4.3 Training and Inference

We load the pre-trained BART parameters except for the standalone vocabulary embeddings for sep-voc model and the extended embeddings for the joint-voc model. We then fine-tune the model with the updated structure-aware architectures on sentence-action pairs with addition of pointer loss.

For decoding, we use similar constrained beam search algorithm as in Zhou et al. (2021), but with our own transition set and rules. We run a state machine on the side to get parser states used by the model. Note that for our joint-voc model, we only allow subword split for node (<string>) actions. As our fine-tuned model is already structure-aware, the graph well-formedness is always guaranteed and no post-processing is needed to return valid graphs, unlike Xu et al. (2020); Bevilacqua et al. (2021). The only post-processing we use is to add wikification nodes as used in all previous parsers.[5]

5 Experimental Setup

Datasets We evaluate our models on 3 AMR benchmark datasets, namely AMR 1.0 (LDC2014T12), AMR 2.0 (LDC2017T10), and AMR 3.0 (LDC2020T02). They have around 10K, 37K, and 56K sentence-AMR pairs for training, respectively.[6] Both AMR 2.0 and AMR 3.0 have wikification nodes but AMR 1.0 does not.

Evaluation We assess our models with SMATCH (F1) scores[7] (Cai and Knight, 2013). We also report the fine-grained evaluation metrics (Damonte et al., 2016) to further investigate different aspects of parsing results, such as concept identification, entity recognition, re-entrancies, etc.

Model Configuration We follow the original BART configuration (Lewis et al., 2019) and code.[8] We use the large model configuration as default, and also the base model for ablation studies. The pointer network is always tied with one head of the decoder top layer, and the pointer loss is added to the model cross-entropy loss with 1:1 ratio for training. Transition alignments are used to mask cross-attention in 2 heads of all decoder layers. For sep-voc model, we build separate embedding matrices for target actions from the training data for decoder input and output space. For joint-voc model, we add new embedding vectors for non-node action symbols and node action strings with a default minimum frequency of 5 (only accounts for about one third of all nodes due to sparsity).

[4] In practice the separate embeddings are initialized with the average subword embeddings from the original BART vocabulary, which gave small gains over random initialization.

[5] We also do light cleaning of the decoded AMR when they are printed to penman format, such as removing reserved characters in node concepts and printing disconnected subgraphs.

[6] See Appendix A for detailed dataset sizes. Data source: https://amr.isi.edu/download.html.

[7] https://github.com/snowblink14/smatch/tree/v1.0.4.

[8] https://github.com/pytorch/fairseq/tree/v0.10.2/examples/bart.

ID	Model	Pre-trained Model	Collapse Subgraph	External Dependency POS	NER	Lemm.	Extra Data	Train Align.	SMATCH F1 (%) AMR 1.0	AMR 2.0	AMR 3.0
1	Naseem et al. (2019)	BERT	S.A.			✓		✓	-	75.5	-
2	Zhang et al. (2019a)	BERT	G.R.	✓	✓				70.2 ±0.1	76.3 ±0.1	-
3	Zhang et al. (2019b)	BERT	G.R.	✓	✓				71.3 ±0.1	77.0 ±0.1	-
4	Cai and Lam (2020)	BERT		✓	✓	✓			74.0	78.7	-
5	Cai and Lam (2020)	BERT	G.R.	✓	✓	✓			75.4	80.2	-
6	Astudillo et al. (2020)	RoBERTa	S.A.			✓		✓	76.9 ±0.1	80.2 ±0.0	-
7	Lyu et al. (2020)	RoBERTa	G.R.		✓	✓			-	-	75.8
8	Bevilacqua et al. (2021)	BART†							-	83.8	83.0
9	Bevilacqua et al. (2021)	BART†	G.R.		✓				-	84.5	80.2
10	Zhou et al. (2021)	RoBERTa	S.A.			✓		✓	78.3 ±0.1	81.7 ±0.1	80.3 ±0.1
11	Xu et al. (2020)	Custom†					4M		-	81.4	-
12	Lee et al. (2020)	RoBERTa	S.A.			✓	85K	✓	78.2 ±0.1	81.3 ±0.0	-
13	Bevilacqua et al. (2021)	BART†					200K		-	84.3	**83.0**
14	Zhou et al. (2021)	RoBERTa	S.A.			✓	70K	✓	-	82.6 ±0.1	-
15	StructBART-S	BART†						✓	81.6 ±0.1	84.0 ±0.1	82.3 ±0.0
16	StructBART-J	BART†						✓	**81.7 ±0.2**	84.2 ±0.1	82.0 ±0.0
17	StructBART-S	BART†					47K	✓	-	-	82.7 ±0.1
18	StructBART-J	BART†					47K	✓	-	**84.7 ±0.1**	82.6 ±0.1
19	StructBART-S ensem.	BART†					47K	✓	-	-	**83.1**
20	StructBART-J ensem.	BART†					47K	✓	-	**84.9**	-

Table 1: SMATCH (%) scores on AMR 1.0, 2.0, and 3.0 test data, associated with each model's dependency on various resources. 1-10/11-14: previous models without/with extra data; 15-18: our models (-S/-J for separate/joint vocabularies for source and target); 19-20: our models with ensemble decoding. Symbols indicate: G.R. - graph re-categorization, S.A. - subgraph action used in transition-based parsers (both detailed in Section 2), POS - part of speech tagger, NER - named entity recognizer, Lemm. - lemmatizer, Align. - alignments (only used at training time). † indicates fine-tuning on top of pre-trained model. All models rely on a external wikification method (ommited). Our results are average of 3 runs with different random seeds. We also report standard deviation and provide ensemble results for the 3 seed combination.

ID	Model	SMATCH	Unlabel	NoWSD	Concepts	NER	Negation	Wiki.	Re-entrancy	SRL
1	Naseem et al. (2019)	75.5	80	76	86	83	67	80	56	72
4	Cai and Lam (2020)	78.7	81.5	79.2	88.1	87.1	66.1	81.3	63.8	74.5
6	Astudillo et al. (2020)	80.2	84.2	80.7	88.1	87.5	64.5	78.8	70.3	78.2
8	Bevilacqua et al. (2021)	83.8	86.1	84.4	90.2	90.6	**74.4**	**84.4**	70.8	79.6
10	Zhou et al. (2021)	81.8	85.5	82.3	88.7	88.5	69.7	78.8	71.1	80.8
15	StructBART-S	84.1	87.5	84.4	90.4	**92.2**	71.0	79.6	73.9	83.0
16	StructBART-J	**84.3**	**87.9**	**84.7**	**90.6**	92.1	72.5	80.8	**74.3**	**83.4**

Table 2: Fine-grained F1 scores on the AMR 2.0 test set, among models that do not use extra silver data and graph re-categorization. The model IDs are matched with those in Table 1 for detailed model features. We report results with our single best model (selected on development data) for fair comparison.

Implementation Details Our models are trained with Adam optimizer with batch size 2048 tokens and gradient accumulation of 4 steps. Learning rate is $1e{-}4$ with 4000 warm-up steps using the inverse-sqrt scheduling scheme (Vaswani et al., 2017). The hyper-parameters are fixed and not tuned for different models and datasets, as we found results are not sensitive within small ranges. We train sep-voc models for 100 epochs and joint-voc models for 40 epochs as the latter is found to converge faster. The best 5 checkpoints based on development set SMATCH from greedy decoding are averaged, and default beam size of 10 is used for decoding for our final parsing scores. We implement our model[9] with the FAIRSEQ toolkit (Ott et al., 2019). More details can be found in the Appendix.

6 Results

Main Results We present parsing performances of our model (StructBART) in comparison with previous approaches in Table 1. For each model, we also list its features such as utilization of pre-

[9]Code and model available at https://github.com/IBM/transition-amr-parser.

Transition System	Features			Model Results on AMR 2.0		Model Results on AMR 3.0	
	#Base Actions	Distant Edges	Special Subgraph	APT* (Zhou et al., 2021)	StructBART sep-voc	APT* (Zhou et al., 2021)	StructBART sep-voc
Astudillo et al. (2020)	12	SWAP	merge	-	-	-	-
Zhou et al. (2021)	10	pointer	merge	81.5 ±0.1	83.4 ±0.1	79.8 ±0.1	81.6 ±0.0
Ours	6	pointer	no	81.6 ±0.1	84.0 ±0.1	79.6 ±0.0	82.3 ±0.1

Table 3: Transition system comparison, including their effects on different parsing models. * we adopt the cited model without graph structure embedding to compare and run on our proposed oracle.

Transition System	Avg. #actions	Oracle SMATCH
Astudillo et al. (2020)	76.2	98.0
Zhou et al. (2021)	41.6	98.9
Ours	45.6	99.9

Table 4: Average action sequence length and oracle coverage on AMR 2.0 training data from different transition systems. Average source sentence length is 18.9.

trained language models and graph simplification methods such as re-categorization. This gives a comprehensive overview of how systems compare in terms of complexity aside from performance.

All recent systems rely on pre-trained language models, either as fixed features or through fine-tuning. The pre-trained BART is particularly beneficial due to its encoder-decoder structure. Among all the models, the graph linearization models (Xu et al., 2020; Bevilacqua et al., 2021) have the least number of extra dependencies when not using graph re-categorization. Our model only requires aligned training data, a trait common to all transition-based approaches. This bears the advantage of producing reliable alignments at decoding time, which are useful for downstream tasks and as explanation of the graph constructing process.

Both our sep-voc and joint-voc model variations work well on all datasets. Without using extra silver data, our model achieves the SMATCH score of 84.2 ±0.1 on AMR 2.0, which is the same as the previous best model (Bevilacqua et al., 2021) with 200K silver data. With the input of only 47K silver data (consisting of ∼20K example sentences of propbank frames and randomly selected ∼27K SQuAD-2.0 context sentences[10]), we achieve the highest score of 84.7 ±0.1 for AMR 2.0. We also attain the high score of 81.7 ±0.2 on the smallest AMR 1.0 benchmark, and the second best score of 82.7 ±0.1 on the largest AMR 3.0 benchmark. Ensemble of the 3 models from the silver training further improves the performances to 84.9 for AMR 2.0 and 83.1 for AMR 3.0.

Fine-grained Results We further examine the fine-grained parsing results on AMR 2.0 in Table 2. We compare models not relying on extra data nor graph re-categorizationn since silver data sets differ across methods, and re-categorization comes with limitations outlined in Section 2. Our models achieve the highest scores across most of the categories, except for negation and wikification. The former may be due to alignment errors and the latter is solved as a separate post-processing step independent of the parser. Compared with the closely related model from Bevilacqua et al. (2021) which also fine-tunes BART but directly on linearized graphs, we achieve significant gains on re-entrancy and SRL (:ARG-I arcs), proving our model generate AMR graphs more faithful to their topological structures.

7 Analysis

Transition System Table 3 and Table 4 compare different transition systems used by recent transition-based AMR parsers with strong performances. Our proposed system has the smallest set of base actions, utilizes the action-side pointer mechanism for flexible edge creation as in Zhou et al. (2021), but does not rely on special treatment of certain subgraphs such as named entities and dates. This results in slightly longer action sequences compared to Zhou et al. (2021), but with almost 100% coverage[11] (Table 4). Our transition system and oracle can always find action sequences with full recovery of the original AMR graph, regardless of graph topology and alignments.

To assess whether our proposed transition system helps integration with pre-trained BART, we train both the APT model from Zhou et al. (2021) and our sep-voc model on the transition system of

[10]https://rajpurkar.github.io/SQuAD-explorer/.

[11]We can recover 100% of AMR 2.0 training graphs excluding 4 with notation errors. Imperfect SMATCH is due to ambiguities of our parser in recovering Penman notation.

Model Variation	SMATCH (%) sep-voc	joint-voc
Ours (hard attention)	84.0 ±0.1	84.2 ±0.1
No align. modeling	83.5 ±0.0	83.4 ±0.2
Align. soft supervision	82.9 ±0.0	83.0 ±0.0
Align. add src emb.	83.9 ±0.0	84.1 ±0.0
No COPY action	83.1 ±0.1	84.1 ±0.0

Table 5: Ablation study of structure modeling with transition alignments. Results are on AMR 2.0 test data.

Zhou et al. (2021) and the one introduced in this work (Table 3 last 4 columns). The APT model, based on fixed RoBERTa features, does not benefit from the proposed transition system. However, our proposed model gains 0.6 on AMR 2.0 and 0.7 on AMR 3.0. This confirms the hypothesis that the proposed transitions are better able to exploit BART's powerful language generation ability.

Structural Alignment Modeling In Table 5, we evaluate the effects of our structural modeling of parser states within BART during fine-tuning. Action-source alignments are natural byproduct of the parser state, providing structural information of where and how to generate the next graph component. Our default use of hard attention to encode such alignments works the best. We explore two other strategies for modeling alignments. One is to supervise cross-attention distributions for the same heads with inferred alignments during training, inspired by Strubell et al. (2018). The other is to directly add the aligned source contextual embeddings from the encoder top layer to the decoder input at every generation step. The former hurts the model performance, indicating the model is unable to learn the underlying transition logic to infer correct alignments, while the latter does equally well as our default model. These results justify the modeling of structural constraints, even when fine-tuning strong pre-trained models such as BART.

We also ablate the use of COPY action in our transition system. The sep-voc model suffers but the joint-voc model is not affected. Without COPY action, the joint-voc model would rely more on BART's pre-trained subword embeddings to split node concepts more frequently, while the sep-vocab model would need to learn to generate more rare concepts from scratch. This indicates that BART's strong generation power is fully used to tackle concept sparsity problems with its subwords.

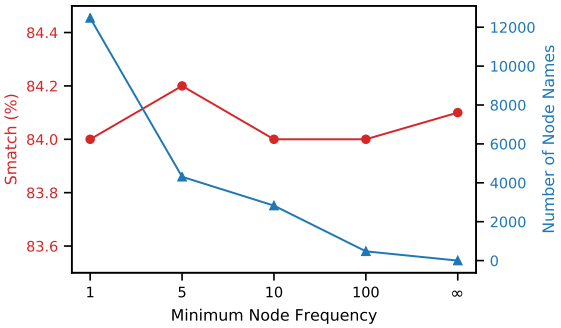

Figure 3: Effect of special AMR node names added to the BART vocabulary in joint-voc model on AMR 2.0 dataset. Remaining AMR concepts are split and generated with BART subwords and sense numbers.

#	Model Initialization src emb	encoder	decoder	tgt emb	SMATCH (%) Base	Large
1					71.2	72.7
2	✓			✓	71.7	72.8
3	✓	✓		✓	81.4	82.8
4	✓		✓	✓	69.2	9.5*
5	✓				71.2	72.8
6	✓	✓			80.9	82.5
7	✓	✓	✓		82.2	83.9
8	✓	✓	✓	✓	82.7	84.1
9	freeze BART src emb				82.6	84.0
10	freeze BART src emb + encoder				80.9	81.8

Table 6: Effects of pre-trained BART parameters. Results are with our sep-voc model on AMR 2.0 data. * failed to converge with a range of hyper-parameters.

Special Nodes in Joint-Voc In Figure 3, we show the joint-voc model performance with different sized (joint) vocabularies. The vocabulary size is controlled by specifying the minimum frequency of occurrence needed for an AMR concept to be added to the vocabulary. For instance, when the minimum frequency is 1, all 12475 AMR concepts from the training data are added onto the BART vocabulary. The number of added concepts decreases as we increase the minimum frequency threshold. On model performance side, we only observe ±0.2 score variations resulting from vocabulary expansion. More interestingly, the model can work equally well when no special concepts are added to the BART vocabulary (minimum node frequency is ∞) – where all the node names are split and generated with BART subword tokens. Although our default setup uses frequency threshold of 5 in joint-voc expansion, following Bevilacqua et al. (2021), it seems unnecessary in terms of achieving good performance. This highlights the efficacy of utilizing the pre-trained BART's lan-

guage generation power for AMR concepts even with relatively small annotated training datasets.

Pre-trained Parameters We study the contribution of different pre-trained BART components in Table 6. With our sep-voc model, we decompose the whole seq-to-seq Transformer into four components for BART initialization, i.e. the source embedding (mapped with BART shared embedding), encoder, decoder, and the separate target embedding (initialized with the average subword embeddings from BART shared embedding). We run both the StructBART base model and the StructBART large model with different combinations of parameter initialization, on the top part of Table 6. We can see that a randomly initialized model of the same size (#1) performs badly. There is an accumulative effect of BART initialization in helping the model performance, except that BART decoder can not work alone well without its encoder (#4). The encoder gives the largest performance gains (#3 vs. #2, #6 vs. #5) of about 10 points. Adding the decoder further gives around 1.4 points on top (#7 vs. #6), justifying its importance as well.

We also experiment with freezing BART parameters during training in the bottom part of Table 6. Our results of freezing the BART encoder are on similar levels of previous best RoBERTa feature based models, which is behind the full fine-tuning. Overall, full initialization from BART with structure-aware fine-tuning (#8) works the best.

8 Related Work

Using seq-to-seq to predict linearized graph sequences for parsing was proposed in Vinyals et al. (2015) and is currently a very extended approach (Van Noord and Bos, 2017; Ge et al., 2019; Rongali et al., 2020). However, it is only recently with the rise of pre-trained Transformer decoders, that these techniques have become dominant in semantic parsing. Xu et al. (2020) proposed custom multi-task pre-training and fine-tuning approach for conventional Transformer models (Vaswani et al., 2017). The massively pre-trained transformer BART (Lewis et al., 2019) was used for executable semantic parsing in Chen et al. (2020) and AMR parsing in Bevilacqua et al. (2021). The importance of strongly pre-trained decoders seems also justified as BART gains popularity in various semantic generation tasks (Chen et al., 2020; Shi et al., 2020). Our work aims at capitalizing on the outstanding performance shown by BART, while providing a more structured approach that guarantees well-formed graphs and yields other desirable sub-products such as alignments. We show that this is not only possible but also attains state-of-the art parsing results without graph re-categorization. Our analysis also shows that contrary to Xu et al. (2020), vocabulary sharing is not necessary for strong performance for our structural fine-tuning.

Encoding of the parser state into neural parsers has been undertaken in various works, including seq-to-seq RNN models (Liu and Zhang, 2017; Zhang et al., 2017; Buys and Blunsom, 2017), encoder-only Transformers (Ahmad et al., 2019), seq-to-seq Transformers (Astudillo et al., 2020; Zhou et al., 2021) and pre-trained language models (Qian et al., 2021). Here we explore the application of these approaches to pre-trained seq-to-seq Transformers. Borrowing ideas from Zhou et al. (2021), we encode alignment states into the pre-trained BART attention mechanism, and re-purpose its self-attention as a pointer network. We also rely on a minimal set of actions targeted to utilize BART's generation with desirable guarantees, such as no unattachable nodes and full recovery of all graphs. We are the first to explore transition-based parsing applied on fine-tuning strongly pre-trained seq-to-seq models, and we demonstrate that parser state encoding is still important for performance, even when implemented inside of a powerful pre-trained decoder such as BART.

9 Conclusion

We explore the integration of pre-trained sequence-to-sequence language models and transition-based approaches for AMR parsing, with the purpose of retaining the high performance of the former and structural advantages of the latter. We show that both approaches are complementary, establishing the new state of the art for AMR 2.0. Our results indicate that instead of simply converting the structured data into unstructured sequences to fit the need of the pre-trained model, it is possible to effectively re-purpose a generic pre-trained model to a structure-aware one achieving strong performance. Similar principles can be applied to adapt other powerful pre-trained models such as T5 (Raffel et al., 2019) and GPT-2 (Radford et al., 2019) for structured data predictions. It is worth exploring thoroughly the pros and cons of introducing structure to the model compared to removing structure from the data (linearization) in various scenarios.

References

Wasi Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. 2019. On difficulties of cross-lingual transfer with order differences: A case study on dependency parsing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2440–2452, Minneapolis, Minnesota. Association for Computational Linguistics.

Ramon Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. *arXiv preprint arXiv:2010.10669*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017a. AMR parsing using stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275, Copenhagen, Denmark. Association for Computational Linguistics.

Miguel Ballesteros and Yaser Al-Onaizan. 2017b. Amr parsing using stack-lstms. *arXiv preprint arXiv:1707.07755*.

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.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. *arXiv preprint arXiv:1704.07092*.

Deng Cai and Wai Lam. 2020. AMR parsing via graph-sequence iterative inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1290–1301, Online. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-resource domain adaptation for compositional task-oriented semantic parsing. *arXiv preprint arXiv:2010.03546*.

Marco Damonte, Shay B Cohen, and Giorgio Satta. 2016. An incremental parser for abstract meaning representation. *arXiv preprint arXiv:1608.06111*.

DongLai Ge, Junhui Li, Muhua Zhu, and Shoushan Li. 2019. Modeling source syntax and semantics for neural amr parsing. In *IJCAI*, pages 4975–4981.

Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramón Fernandez Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, Dinesh Garg, Alfio Gliozzo, Sairam Gurajada, Hima Karanam, Naweed Khan, Dinesh Khandelwal, Young-Suk Lee, Yunyao Li, Francois Luus, Ndivhuwo Makondo, Nandana Mihindukulasooriya, Tahira Naseem, Sumit Neelam, Lucian Popa, Revanth Gangi Reddy, Ryan Riegel, Gaetano Rossiello, Udit Sharma, G P Shrivatsa Bhargav, and Mo Yu. 2021. Leveraging Abstract Meaning Representation for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3884–3894, Online. Association for Computational Linguistics.

Young-Suk Lee, Ramon Fernandez Astudillo, Tahira Naseem, Revanth Gangi Reddy, Radu Florian, and Salim Roukos. 2020. Pushing the limits of amr parsing with self-learning. *arXiv preprint arXiv:2010.10673*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Michael Lingzhi Li, Meng Dong, Jiawei Zhou, and Alexander M Rush. 2019. A hierarchy of graph neural networks based on learnable local features. *arXiv preprint arXiv:1911.05256*.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A Smith. 2018. Toward abstractive summarization using semantic representations. *arXiv preprint arXiv:1805.10399*.

Jiangming Liu and Yue Zhang. 2017. Encoder-decoder shift-reduce syntactic parsing. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 105–114, Pisa, Italy. Association for Computational Linguistics.

Chunchuan Lyu, Shay B Cohen, and Ivan Titov. 2020. A differentiable relaxation of graph segmentation and alignment for amr parsing. *arXiv preprint arXiv:2010.12676*.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. Association for Computational Linguistics.

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Tahira Naseem, Srinivas Ravishankar, Nandana Mihindukulasooriya, Ibrahim Abdelaziz, Young-Suk Lee, Pavan Kapanipathi, Salim Roukos, Alfio Gliozzo, and Alexander Gray. 2021. A semantics-aware transformer model of relation linking for knowledge base question answering. ACL.

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 the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4586–4592, Florence, Italy. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Peng Qian, Tahira Naseem, Roger Levy, and Ramón Fernandez Astudillo. 2021. Structural guidance for transformer language models. 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)*, pages 3735–3745, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of The Web Conference 2020*, pages 2962–2968.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020. Learning contextual representations for semantic parsing with generation-augmented pre-training. *arXiv preprint arXiv:2012.10309*.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. *arXiv preprint arXiv:1804.08199*.

Rik Van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in neural information processing systems*, pages 2773–2781.

Andreas Vlachos et al. 2018. Guided neural language generation for abstractive summarization using abstract meaning representation. *arXiv preprint arXiv:1808.09160*.

Chuan Wang and Nianwen Xue. 2017. Getting the most out of amr parsing. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1257–1268.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020. Improving amr parsing with sequence-to-sequence pre-training. *arXiv preprint arXiv:2010.01771*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. Amr parsing as sequence-to-graph transduction. *arXiv preprint arXiv:1905.08704*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3784–3796, Hong Kong, China. Association for Computational Linguistics.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, and Radu Florian. 2021. Amr parsing with action-pointer transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5585–5598, Online. Association for Computational Linguistics.

A Dataset Statistics

We list the dataset sizes of AMR benchmarks in Table 7. The sizes increase with the release version number. AMR 2.0 is the most used by far. AMR 2.0 shares the same set of sentences for development and test data with AMR 1.0, but with revised annotations and wikification links. AMR 3.0 is released most recently, which is under-explored.

Our silver data originate from two sources. First, we use ∼20K example sentences (∼386K tokens) from the propbank frames included in the AMR 3.0 distribution. Second, we use randomly selected ∼27K sentences (∼620K tokens) from SQuAD 2.0 context sentences available from https://rajpurkar.github.io/SQuAD-explorer/.

Data Split	AMR 1.0	AMR 2.0	AMR 3.0
Training	10,312	36,521	55,635
Development	1,368	1,368	1,722
Test	1.371	1.371	1,898

Table 7: Number of sentence-AMR instances in the AMR benchmark datasets.

B Details of Model Structures and Number of Parameters

In Table 8, we list the detailed model configuration and number of parameters of the official pre-trained BART models. Our fine-tuned StructBART is with different action vocabulary strategies which builds additional embedding vectors for certain action symbols. The numbers vary from training dataset. We list the detailed number of parameters of our fine-tuned model in Table 9. The fine-tuned model only increases about 3%-8% more parameters for sep-voc model and 0.4%-1% more parameters for joint-voc model.

C Implementation Details

We use the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Batch size is set to 2048 maximum number of tokens, and gradient is accumulated over 4 steps. The the learning rate schedule is the same as Vaswani et al. (2017), where we use the maximum learning rate of $1e{-}4$ with 4000 warm-up steps. Dropout of rate 0.2 and label smoothing of rate 0.01 are used. These hyper-parameters are fixed and not tuned for different models and datasets, as we found results are not sensitive within small ranges. Without silver data, we train sep-voc models for 100 (AMR 1.0 & 2.0) or 120 (AMR 3.0) epochs and joint-voc models for 40 epochs as the latter is found to converge faster. The best 5 (AMR 1.0 & 2.0) or 10 (AMR 3.0) checkpoints among the last 40/30 epochs are selected based on development set SMATCH from greedy decoding and averaged over the model parameters as our final model. With the 50K silver data, we train both sep-voc and joint-voc models for 20 epochs and select the best 10 checkpoints for model parameter averaging. We use a default beam size of 10 for decoding for our final parsing scores. Our models are implemented with the FAIRSEQ toolkit (Ott et al., 2019), trained and tested on a single Nvidia Tesla V100 GPU with 32GB memory. We use fp16 mixed precision training whenever possible, with which training a large model on AMR 2.0 takes about 10 hours for sep-vocab models and 7 hours for joint-vocab models, and the time varies proportionally with data size for other datasets and with silver data.

Configuration	BART Base	BART Large
Encoder layers	6	12
Heads per layer	12	16
Hidden size	768	1024
FFN size	3072	4096
Size of vocab	51201*	50265
Size of emb. matrix	39,322,368	51,471,360
#parameters trained	140,139,266	406,291,458

Table 8: Original model configurations of pre-trained BART from FAIRSEQ (https://github.com/pytorch/fairseq/tree/v0.10.2/examples/bart). The embeddings for source, decoder input and output are all shared and thus the same (not counted as extra in training parameters). *vocabulary for the base model is larger due to additional paddings at the end, but effective vocabulary symbols are the same as the large model.

Model	Param.	AMR 1.0	AMR 2.0	AMR 3.0
sep-voc	Src vocab size	50265	50265	50265
	Tgt vocab size	6976	12752	16180
	#param. trained	420,578,304	432,407,552	439,436,288
joint-voc	joint vocab size	51921	53487	54388
	#param. trained	407,987,200	409,590,784	410,517,504

Table 9: Model parameters of our StructBART (large model). The sizes differ based on the target side vocabulary, which is dependent on different training data. Addition of silver training data adds only a fraction of the parameters to the benchmark datasets.