End-to-end Parsing of Procedural Text into Flow Graphs

Ahmad Pourihosseini[1,3*†], **Dhaivat Bhatt**[1*], **Federico Fancellu**[1,2†], **Afsaneh Fazly**[1]
[1]Samsung AI Centre Toronto, [2]Solventum, [3]University of Toronto, [*]Equal contribution
{d.bhatt, a.fazly}@samsung.com
a.pourihosseini@gmail.com, ffancellu@solventum.com

Abstract

We focus on the problem of parsing procedural text into fine-grained flow graphs that encode actions and entities, as well as their interactions. Specifically, we focus on parsing cooking recipes, and address a few limitations of existing parsers. Unlike SOTA approaches to flow graph parsing that work in two separate stages — identifying actions and entities (tagging) and encoding their interactions via connecting edges (graph generation) — we propose an end-to-end multi-task framework that simultaneously performs tagging and graph generation. In addition, due to the end-to-end nature of our proposed model, we can unify the input representation, and moreover can use compact encoders, resulting in small models with significantly fewer parameters than SOTA models. Another key challenge in training flow graph parsers is the lack of sufficient annotated data, due to the costly nature of the fine-grained annotations. We address this problem by taking advantage of the abundant unlabelled recipes, and show that pre-training on automatically-generated noisy silver annotations (from unlabelled recipes) results in a large improvement in flow graph parsing.

Keywords: Procedural text understanding, End-to-end flow graph parsing, Silver data pre-training

1. Introduction

Activities involving the use of procedural content are abundant in our daily lives, e.g., following a cooking recipe or a furniture assembly video. Intelligent assistants can help guide users follow instructions in a procedure to accomplish a task. A fundamental requirement for building systems that help users follow a procedure, such as a cooking recipe, is the ability to understand and represent the dynamics of the underlying process, including the actions and participating entities, as well as how they interact and affect each other. Due to the abundance of cooking recipe/video data, and its widespread application, much work on procedure understanding has focused on the cooking domain. Rich and structured representations of procedural (cooking) content have been shown to enable an array of downstream tasks, including cooking recipe search (Xie et al., 2010), retrieval (Wang et al., 2008), or recommendation (Lo et al., 2015), as well as recipe-to-video alignment in a multi-modal setting (Dvornik et al., 2022).

One important aspect of procedure understanding is learning to parse a recipe text into a *flow graph* that encodes the actions and entities, as well as the overall step-by-step process; see Figure 1 for an example. Much work has focused on learning such graphs from cooking recipes (Xie et al., 2010; Walter et al., 2011; Jermsurawong and Habash, 2015; Kiddon et al., 2015; Pan et al., 2020; Yamakata et al., 2020; Donatelli et al., 2021). Early work has built complex fine-grained cooking flow graphs by relying on linguistic knowledge and/or hand-crafted rules (Xie et al., 2010; Walter et al., 2011; Kiddon et al., 2015). To address the limited scalability of such methods, others focused on the automatic generation of flow graphs. Whereas some only focus on specific actions within these, e.g., ingredient–instruction dependencies (Jermsurawong and Habash, 2015) and instruction-level temporal relations (Pan et al., 2020), more recent work (Yamakata et al., 2020; Donatelli et al., 2021) uses machine learning techniques to generate all elements of a flow graph from recipe texts. These models rely on supervised approaches that require rich manually annotated recipes that are costly to acquire. Consequently, such datasets are often very small, and thus impose limitations on the general applicability of the parsers trained on them. In addition, these models draw on separate systems to first identify and tag the entities in a recipe, and then connect them via relations to form the final graph. This two-stage approach makes the models less robust to tagging errors during inference. Finally, current evaluations are limited to a small test set, making it hard to draw conclusions about real-world performance.

In this paper, we address the above-mentioned limitations in existing flow graph parsing. Specifically, we experiment with end-to-end architectures and small encoders that result in models with significantly fewer parameters than SOTA models. In addition, we draw on the use of automatically-generated (noisy) training data to avoid the high cost of generating fine-grained annotations required by SOTA models. Finally, we propose a

[†]Work done while at Samsung AI Centre Toronto.

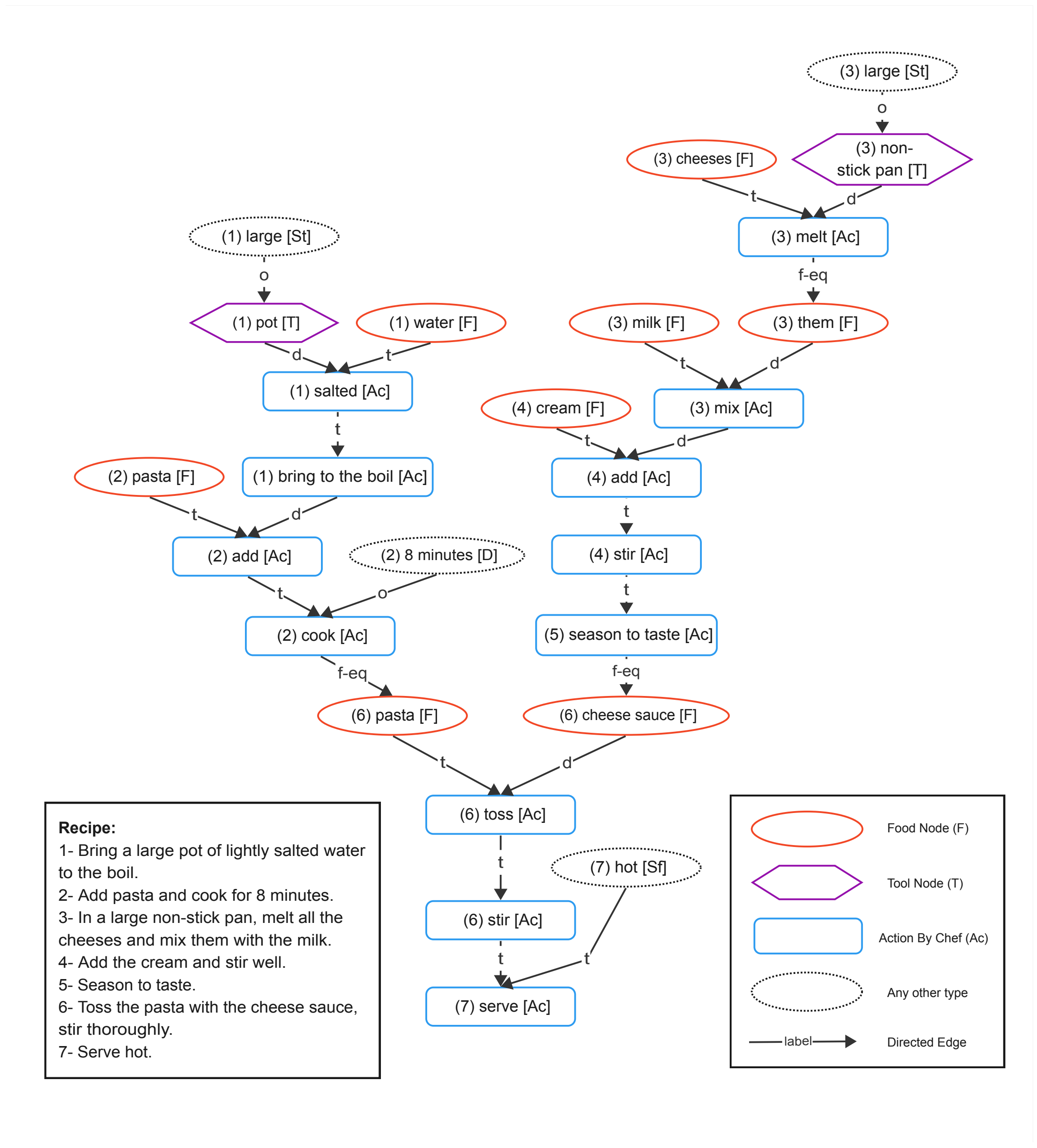

Figure 1: A sample recipe and its flow graph from Yamakata et al. (2020)'s dataset. Nodes encode cooking entities, e.g., Action and Food. Edges encode pairwise relations among nodes, e.g., *cheese* is the "target" (t) of the *melt*, and *non-stick pan* is its "destination" (d).

new evaluation framework that overcomes the sensitivity of current evaluations to the composition of the test set. The following paragraphs describe our contributions in more detail.

End-to-end Architecture. Previous work on flow graph parsing draws on two separate systems to identify named entities in a recipe, and to connect them via relations to form the final graph. We instead build an end-to-end architecture where decisions made in either (entity tagging or graph generation) task can inform the other. Because of the end-to-end architecture, we can also simplify the input representation by deploying a single set of pre-trained word embeddings as input features. We show that the performance of our model is comparable to previous models while having significantly fewer parameters and lower complexity. Additionally, we show that we can keep competitive performance while further reducing the model size by around 70% by using a light-weight encoder. In addition, our end-to-end model is more robust to tagging errors since it is trained on its own tag predictions (in contrast to SOTA model trained with ground-truth tags).

Addressing Data Scarcity. We hypothesize that the small size of annotated training data could impact performance. To alleviate this problem, we follow work in semantic parsing that uses automatically generated silver data to address data scarcity (e.g., Van Noord and Bos, 2017; Xu et al., 2020; Tedeschi et al., 2021; Xia et al., 2021). Specifically, we use an existing flow graph parser (Donatelli et al., 2021) to parse a large set of unlabelled recipes, which we use as additional (noisy) training data for our end-to-end model. Our results show that using silver data yields an average performance improvement of around 6–8 F1 points.

Comprehensive Evaluation. We conduct a thorough evaluation of existing and new models. We show that due to the small size of existing evaluation sets, performance can vary greatly on different recipes. We propose a new way of evaluation to address this sensitivity to the composition of test data. Finally, we perform ablation studies and error analysis to understand how performance is affected by different factors.

We believe the above contributions will help establish standard frameworks for data generation, modeling, and evaluation of flow graph parsers for procedural understanding.

2. End-to-end Flow Graph Parsing

We discuss our architecture in Section 2.1, elaborating on our contributions over SOTA. We then discuss our motivation behind the use of the SOTA model to generate silver data, and how this addresses data scarcity.

2.1. Unified Architecture

Our end-to-end model is based on several architectural changes to the state-of-the-art method of Donatelli et al. (2021), where we combine the two stages of entity tagging and graph generation into a single pipeline, effectively jointly learning to tag and parse an input recipe. By doing so, we draw on multi-task learning and simultaneously learn to tag and parse an input recipe, using a single input embedding. This is in contrast to the model of Donatelli et al. in which the tagger and parser draw on different embeddings, and are completely independent of each other. A high-level comparison of the two architectures is shown in Figure 2. The following paragraphs elaborate on our modifications to the model of Donatelli et al..

Simplified input representation. We use the same BERT embedding for both tagging and parsing.[1] Specifically, we encode a recipe with n words as $r_{1:n}$, where r_i is a word-level embedding taken from BERT (Devlin et al., 2018) subword representations via averaging. This is in contrast to the input layer of Donatelli et al. where the input recipes are encoded separately in the tagger and the parser using a combination of different character and word embeddings.

Reduced tagger complexity. We reduce the complexity of the tagger by removing the CRF layer. Recall that the tagger assigns each word a cooking-related tag (e.g., action, food, etc.). Similar to Donatelli et al., we use a BiLSTM with an MLP layer on top to obtain tag logits $s_{1:n}$, but we remove the additional CRF layer, while yielding better performance and reducing computational complexity. The CRF classifier has an additional runtime complexity of $\mathcal{O}(nK^2)$, where K is the number of possible tag labels, which is 21 in our case (considering the BIO tagging scheme), whereas the MLP classifier has $\mathcal{O}(n)$ complexity.

Joint tagging and parsing. We unify the tagger and parser module via directly feeding the tagger output as vectors into the parser. To do this, we first transform the $s_{1:n}$ logits from the tagger into near-one-hot representations via a softmax function with a low temperature (i.e., softargmax or a differentiable argmax.) to obtain sparse tag vectors. We then pass these learned tag vectors through a feed-forward layer to form dense representations that are concatenated with the recipe encoding $r_{1:n}$ to serve as the input to the parsing module.

Our parsing module is the same as that of Donatelli et al., which is essentially the Biaffine parser of Dozat and Manning (2016). The Biaffine parser consists of three main components: a BiLSTM module that enchances the input representations, a Biaffine scorer that scores the presence of an edge as well as its label for every pair of nodes, and a Maximum Spanning Tree algorithm that generates the output based on the learned edge scores.[2]

Our tagger uses the standard cross-entropy loss between predicted and ground-truth tags. The parser draws on two separate cross-entropy objectives: (i) to predict the presence of an edge between each pair of nodes; and (ii) to predict an edge label should it exist. These three objectives are used to jointly train our end-to-end model.

2.2. Silver Data Pre-training

A major limitation for fine-grained flow graph parsing is the lack of sufficiently large annotated data. Recent work in semantic parsing and entity tagging has shown that pre-training on a large corpus of automatically generated noisy *silver* data

[1] Initial experiments with a larger text encoder, BART (Lewis et al., 2019), did not show improvements over BERT, and hence we did not conduct further experiments with BART.

[2] Note that due to the application of the MST algorithm, predicted flow graph outputs are actually (inverse) trees.

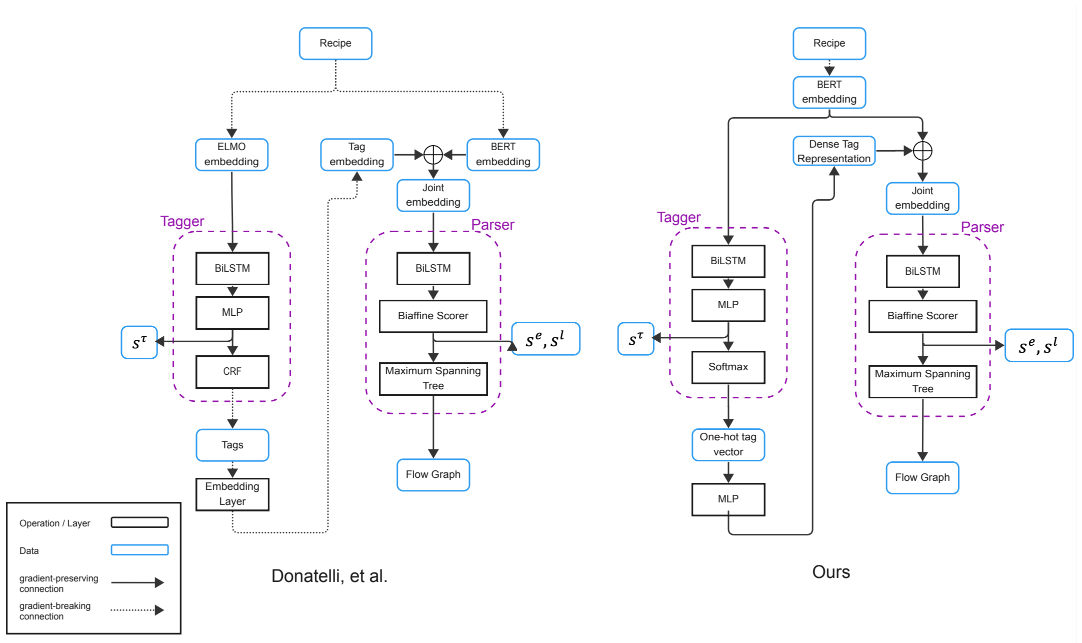

Figure 2: Comparison between our proposed architecture and that of Donatelli et al. (2021). The straight, black rectangles represent operations, while the rounded rectangles represent data. Solid arrows are connections that preserve the gradient chain, and dashed arrows are those that break it.

is beneficial (Van Noord and Bos, 2017; Xu et al., 2020; Tedeschi et al., 2021; Xia et al., 2021). Following these, we automatically annotate a large collection of unlabelled recipes using the model of Donatelli et al. (2021) to be used as silver data for pre-training.

For a model to benefit from silver data, it should be sufficiently different from the model used for generating the data. We believe this is true for our case for the following reasons:

- There are several important differences between our model and that of Donatelli et al., including an end-to-end architecture, unified input embedding, and the use of predicted tags vs. ground-truth tags for training the parsing module.

- The final flow graph predictions of Donatelli et al.'s model are produced by applying a Maximum Spanning Tree (MST) algorithm on top of predicted edge scores. We thus expect these to provide additional supervisory signal because they are different from the output of the greedy decoding used during training.

3. Experimental Setup

Gold data. We use the English Recipe Flow Graph corpus (Yamakata et al., 2020), consisting of 300 annotated recipes. In this dataset, recipe instructions are annotated with a set of 10 cooking entity tags, including Ac(tion), F(ood), and T(ool). These entities effectively form the nodes of the flow graph, connected via edges whose labels specify the relation between a pair of nodes. Further details on the annotation can be found in the original paper. An example is shown in Fig. 1. Unless stated otherwise, we use the train/validation/test splits introduced by Donatelli et al. (2021), referred to as D21.[3]

Silver data. For silver data, we sample 10K recipes from the Recipe1M dataset (Marin et al., 2019), such that their distribution in terms of length (number of tokens) resembles that of D21. We choose recipes whose length is within two standard deviations of the average recipe length in D21 ($130 \pm 2 \times 71$), excluding short recipes (shorter than 20 tokens). These recipes are automatically parsed using the model of Donatelli et al. (2021), and are used to pre-train our model. Section 5.1 presents an ablation over the size of silver training data to explain why we selected 10K recipes. We split the 10K recipes into 80% training, 10% validation, and 10% test.

Evaluation. Following prior work (Yamakata et al., 2020; Donatelli et al., 2021), we report Precision, Recall, and F1 for tagging and parsing. For parsing, we noted that while the results reported in Yamakata et al. consider edge labels (labelled

[3]The released D21 dataset is missing 3 of the original recipes, containing a total of 297 recipes, divided into 238 training, 30 validation, and 29 test recipes.

Model	Training Data	P	R	F1
Donatelli (2021)	original D21	74.4	70.4	72.3
Donatelli$^{\dagger b}$	original D21	74.2	69.5	71.8
Donatelli$^{\dagger b}$	corrected D21	72.8	68.4	70.5

Table 1: Unlabelled parsing results on D21 test; † marks re-produced results; ^{b} indicates best results out of 20 runs. Training data is either original, or corrected.

			Parser	
Model	Size	Tagger	Labelled	Unlabelled
Donatelli$^{\dagger b}$	294M	86.3	61.5	70.5
Ours	129M	88.3	68.2	76.7
Ours (LW)	40M	86.7	67.2	76.0

Table 2: Comparing F1 of our original and light-weight (LW) models on D21 test, trained with silver+gold. Donatelli et al.'s best re-produced results are also given.

edge prediction), those in Donatelli et al. do not (confirmed with the authors). Labelled edge prediction considers the labels assigned to edges to determine correctness, whereas unlabelled prediction only considers the presence or absence of an edge between a pair of nodes. For completeness, we report both labelled and unlabelled edge prediction results. Additionally, the two studies use different evaluation schemes: whereas Yamakata et al. report results using 10-fold cross-validation, Donatelli et al. (D21) split the dataset into 80% training, 10% validation, and 10% test, and report results on the test set. In our evaluation, we observed a notable gap (around 7–8 F1 point) between performances on validation and test portions. Thus, in addition to reporting results on the D21 test set, we report average and standard deviations over 30 runs of each model, where each run uses a different random 80%/10%/10% train/validation/test split. We believe this is a better evaluation scheme, compared to cross-validation or single test split, because of the very small size of the data.

Implementation details. We implement our models using PyTorch, AllenNLP, and Transformers libraries (Paszke et al., 2017; Gardner et al., 2017; Wolf et al., 2019). For re-producing the results of Donatelli et al. (2021), we use their implementation.[4]. We use a batch size of 16 for silver data training and a batch size of 8 for training on gold data (D21). When exclusively trained on silver data or D21 data, the models are trained for 40 epochs and 80 epochs, respectively. When first trained on silver data and then fine-tuned on D21 data, the model is trained for 80 epochs and then fine-tuned for 100 epochs. All models are trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 1e-3, using labelled precision as the early stopping criterion. Note that we use the D21 or 1K silver validation sets for early stopping. We use the pre-trained bert-base-uncased (Wolf et al., 2019), and keep it frozen during training. For the light-weight model, we use the 6-layered version of MiniLM (Wang et al., 2020). Temperature parameter for softargmax is fixed at 1e-3 in all experiments. We performed all the experiments on a single Nvidia A6000 GPU with 48GB memory.

[4]https://github.com/interactive-cookbook/ara

4. Results

We perform an extensive set of experiments to understand the effect of our end-to-end architecture and a light-weight encoder, as well as the use of silver data for training. In addition, we empirically show that a multi-split evaluation framework is more suited for the task of flow graph parsing given the small size of the existing evaluation data.

Re-producing existing SOTA. Our goal is to perform a comprehensive analysis of Donatelli et al. (2021)'s parser, as well as our end-to-end model. Donatelli et al. (2021) report the performance of their parser on D21 test (with predicted and ground-truth entity tags). We first aim to re-produce the results of Donatelli et al., focusing on predicted tags only, since we assume no access to ground-truth tags at inference time.[5] Table 1 shows reported and re-produced results of Donatelli et al. under two settings of the training data: original, and corrected, where two erroneous labels are replaced by the correct ones, in consultation with the data creators.[6] This correction, although necessary to clean the data, results in some drop in performance. We perform all our experiments with

[5]We re-produce the results of Donatelli et al. using their released code, so we can run their model on other data splits and settings.

[6]Labels *s* and *v* are replaced by *d* and *v-tm*, respectively.

	Labelled			Unlabelled		
Model	P	R	F1	P	R	F1
Donatelli$^{\dagger b}$T	63.5	59.7	61.5	72.8	68.4	70.5
Donatelli$^{\dagger b}$V	71.3	67.7	69.5	80.5	76.4	78.4
Ours^{b}T	64.4	61.9	63.1	72.7	69.8	71.2
Ours^{b}V	72.9	69.8	71.3	80.7	77.2	78.9

Table 3: Parsing results (best out of 20 runs) on D21 T(est) and V(alidation); † marks re-produced results.

Model	Training	Tagger	Labelled	Unlabelled
Ours^{b}	gold	87.5	63.1	71.2
Ours	silver	88.0	66.6	75.6
Ours	silver + gold	**88.3**	**68.2**	**76.7**

Table 4: Tagging and parsing F1 on D21 test, using different combinations of gold and silver data for training.

the correction. When reporting best results out of several runs, each run uses a different random seed and the labelled edge prediction F1 on validation is used to find the best model.

Our end-to-end parser performance. Table 3 reports results of our model as well as that of Donatelli et al., when trained and tested on the D21 splits. Note that the reported results of Yamakata et al. (51.1, 37.7, 43.2 for P, R, and F1, respectively, for labelled edge prediction) are not comparable to those in Table 3 since they were done using 10-fold cross-validation (folds are not available for experimentation). We observe a small advantage for our model (between .5 and 1.8 in F1). However, both models show a large performance difference (around 8 F1 points) between validation and test results. We return to this issue later.

Effect of silver data pre-training. Table 4 shows the benefit of pre-training our model with silver data followed by fine-tuning on gold, showing more than 5 points increase in parsing F1, compared to the best model trained with gold only. Compared to the baseline model, this shows an 8.8% relative improvement (76.7 vs. 70.5). Importantly, this advantage is not due to a better tagger (tagging performance is comparable), suggesting that the benefit is likely due to better edge prediction by the parser.

Evaluation using multiple splits. We observed a large performance gap between validation and test results (Table 3). To alleviate this sensitivity to dataset split, we design an experiment where we use 30 different random splits of D21 for training and evaluation, where each split is composed of 80% training, 10% test, and 10% validation recipes. Table 5 reports average and standard deviation over the 30 runs, demonstrating the superiority of our model when pre-trained on silver and fine-tuned on gold data, resulting in an improvement of around 6–8 F1 points. Importantly, standard deviations are small for all models and training sets, suggesting that the performance gap between the different splits is rather small. We argue that for a small dataset, performing experiments over multiple splits provides a better evaluation scheme and removes sensitivity to a particular split.

Using a light-weight text encoder. We use the 6-layered version of MiniLM (Wang et al., 2020) as our light-weight encoder (replacing BERT in Fig. 2 (ours).). Table 2 reports the tagging and parsing results for the original and the light-weight (LW) version of our model. Interestingly, we observe a very small drop in performance, with a significant reduction in model size of around 70%. When compared to the Donatelli et al.'s model, Ours (LW) has a superior performance ($5+$ points increase in parsing F1) with 86% fewer parameters.

5. Ablations and Error Analysis

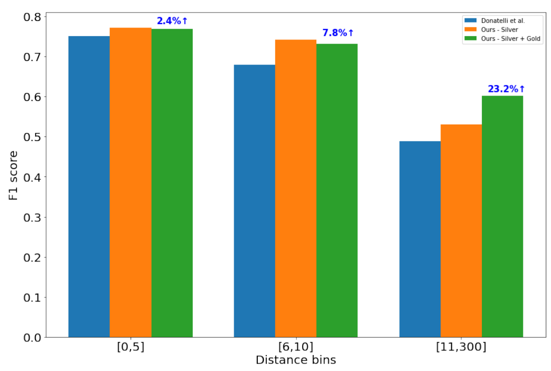

Figure 3: Distance-specific F1 (on D21 validation set) for the models of Donatelli et al. and Ours trained with either silver data, or silver+gold.

Model	Training	Labelled	Unlabelled
Donatelli[†]	gold	67.9 ± 1.8	75.9 ± 1.6
Ours	gold	66.2 ± 2.0	73.9 ± 2.3
Ours	silver	75.5 ± 2.0	81.5 ± 1.7
Ours	silver + gold	$\mathbf{75.7 \pm 1.9}$	$\mathbf{81.7 \pm 1.7}$

Table 5: Parsing F1 (average $\pm$ standard deviation) over 30 random test splits of the D21 data.

Edge Label	Gold Data	Silver Data
t	41.2%	42.6%
o	18.6%	17.6%
d	13.9%	15.2%
f-eq	6.3%	7.1%
t-comp	4.3%	4.2%
v-tm	4.3%	4.2%
a	3.7%	3.0%
f-part-of	2.5%	1.7%
f-comp	2.0%	2.3 %
a-eq	1.2%	0.5%
t-eq	1.1%	0.7%
t-part-of	1.0%	0.9%
f-set	0.1%	0.0%
Total	11.8K	357.2K

Table 6: Relative frequency of each edge label in the gold and silver training sets, sorted by frequency in gold data. The last row shows the total number of edges in each dataset.

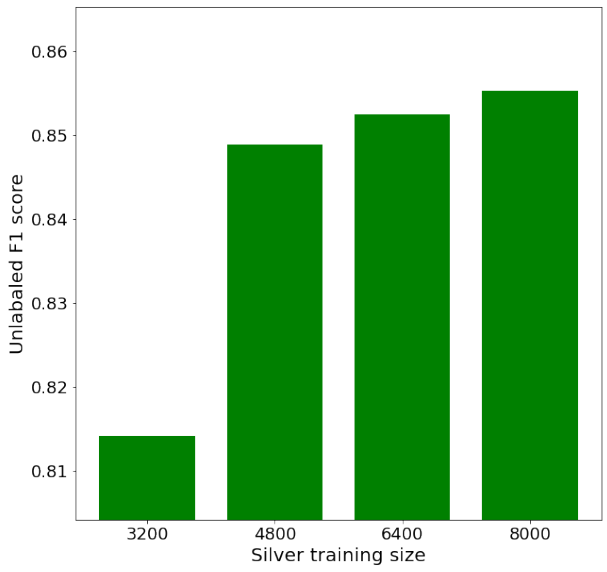

Figure 4: Unlabelled F1 score for models trained with different silver data training sizes, from 3.2K to 8K, tested with the same 1K silver test data.

In this section, we perform some error analyses to understand the role of an end-to-end versus a two-stage model. We also look at the breakdown of performance for different relations, as well as different head-dependent distances.

5.1. Optimal Size of Silver Data

To determine the optimal size of silver data, we train our model with different silver training data sizes (from 3.2K to 8K) and evaluate on 1K silver test recipes. We show unlabaled F1 score for these different models in Figure 4. We observe that performance starts to saturate despite the increase in training size. We set the silver data size to 8K.

5.2. Cascading Errors

The baseline model (Donatelli et al., 2021) consists of a tagger and a biaffine parser component, where the predictions of the parser build on top of the predictions provided by the tagger. One of the critical weaknesses of systems with multiple components is that errors made in an earlier stage cascade to the following decisions. In addition, the baseline model's parser is trained with the gold tags, which can lead to the exposure bias problem when transitioning to predicted tags during inference.

Table 7 shows the percentage of parsing errors made by the baseline and our models, when the tag prediction is correct vs. when it is wrong. When comparing Ours (gold) with the baseline (first row), we see a slight increase in errors with correct tags, but a sharp decline in errors when the tags are wrong. This is probably due to the fact that our model is being trained using predicted tags. So, the parser knows how to handle an incorrect tag prediction better, but does worse on correct tags since it has seen noisy tag inputs during training. When using silver data for pre-training, we see a similar decline in parsing errors when tags are predicted incorrectly, and slightly lower error rate for correctly predicted tags. Overall, the model that uses both silver and gold data has the lowest rate of parsing errors. This shows that pre-training on silver data and fine-tuning on gold data results in successfully reducing the parser's dependence on correct tag predictions, hence a robust model.

5.3. Performance by Relation Type

Figure 5 shows F1 scores for labelled edge prediction, broken down by each relation type (edge label). Labels are sorted by frequency (based on gold training data) from left to right, with their relative frequencies in the silver and gold training

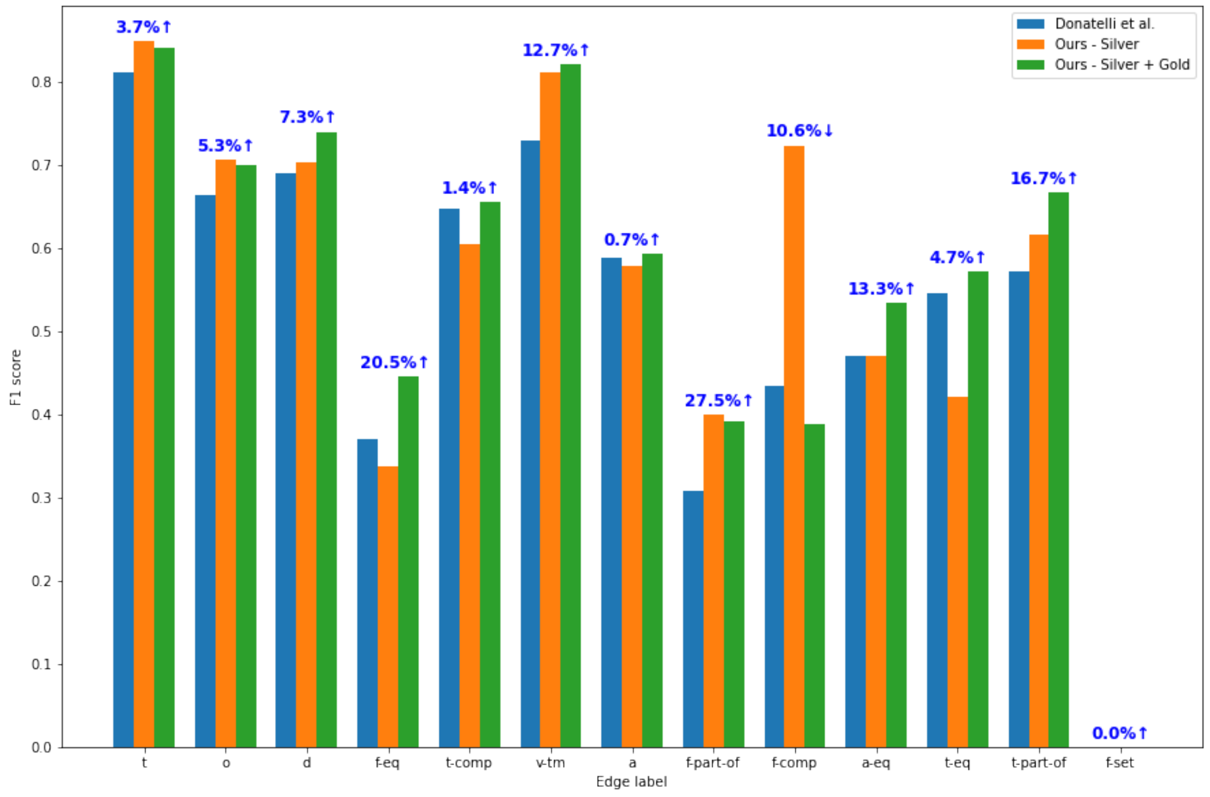

Figure 5: Label-specific F1 (on D21 validation set) for the models of Donatelli et al. and Ours trained with either silver data, or silver+gold.

Model	Training	Wrong Tag	Correct Tag
Donatelli$^{\dagger b}$	gold	70.3%	10.1%
Ours	gold	56.2%	10.5%
Ours	silver	59.0%	9.2%
Ours	silver+gold	53.1%	9.3%

Table 7: Percentage of parsing errors (on D21 validation set) separated by whether the tag prediction is correct or wrong. A prediction is considered to contain a parsing error if either the head prediction or edge label prediction is wrong.

sets given in Table 6. We observe that the best overall model (Ours trained with silver + gold) has slightly better performance than the baseline for the most frequent labels (t, o and d) that form more than 75% of the labels in the training data. For the nine labels with lower frequency, we observe large relative performance improvement for six of them (f-eq, v-tm, f-part-of, a-eq, t-eq, and t-part-of), comparable performance for two (t-comp, a), and a notable decline in performance for one (f-comp). This overall improvement could simply be the result of having more data to learn from when training on (noisy) silver data, especially demonstrating the benefits of pre-training for low frequency items. Although the decline in performance for the one label requires further investigation.

5.4. Effect of Head–Dependent Distance

We perform an experiment to understand how distance between a head token and its dependent affects the parsing performance. Figure 3 shows parsing F1 for three models (baseline model of Donatelli et al. (2021) as well as Ours trained with silver or silver + gold) for three head–dependent distance bins. E.g., the bin $[0, 5]$ contains head–dependent token pairs that are $\leq$ 5 words away from each other in the original recipes. As expected, we can see that as distance increases, parsing performance goes down for all three models. However, our best model (trained on silver + gold) shows more advantage over the baseline for larger distances. Especially, our best model has a relative improvement of 23.2% (over the baseline) for the most distant pairs (the bin corresponding to $[11, 300]$). These results further demonstrate the benefit of using silver data for hard cases.

6. Limitations

Our experiments are limited to English data. Further experiments are needed to validate the applicability of our methods to other languages. Comparisons are performed against the most recent

SOTA only, due to unavailability of other models.

7. Conclusions

We addressed several key challenges in flow graph parsing of cooking recipes. In particular, we proposed ways of automatically generating additional training data, frameworks for better evaluation, and a simple unified end-to-end architecture for building flow graphs from recipes. We introduced an end-to-end model for jointly tagging and parsing recipes into flow graphs that outperformed previous approaches, made use of silver data to further boost performance, and could be made lightweight with negligible loss in performance. Finally, we suggested a new evaluation framework to alleviate a problem we observed in using existing small test data for evaluation. Our goal has been to establish standard frameworks for data generation, modeling, and evaluation of flow graph parsers, in order to help the community make progress on the task of flow graph parsing from procedural text. Future work will look into unifying aspects of flow graphs with other graph-based semantic formalisms, such as abstract meaning representations (Banarescu et al., 2013).

8. Acknowledgement

A. Pourihosseini wishes to express gratitude to his academic supervisor, Frank Rudcicz, for the helpful feedback throughout the development of this work, conducted during his Master's studies, and to the Vector Institute for supporting his studies through the Vector Scholarship in Artificial Intelligence.

9. Ethical Considerations

In this paper, we proposed an end-to-end model for recipe flow graph parsing. Our model uses large transformer-based pre-trained language models, which is known for having significant GPU/memory/compute footprint due to prolonged training schedules. Our work builds on top of such pre-trained models. Importantly, we show that by using a smaller scale language model, we can achieve state of the art parsing performance, while significantly reducing model size. Compared to prior art, unifying the two tasks (tagging and parsing) into a single pipeline eliminates the need for redundant computational blocks, thus greatly reducing memory and compute requirements.

10. References

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Lucia Donatelli, Theresa Schmidt, Debanjali Biswas, Arne Köhn, Fangzhou Zhai, and Alexander Koller. 2021. Aligning actions across recipe graphs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6930–6942.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Nikita Dvornik, Isma Hadji, Hai Pham, Dhaivat Bhatt, Brais Martinez, Afsaneh Fazly, and Allan D Jepson. 2022. Graph2vid: Flow graph to video grounding forweakly-supervised multi-step localization. *arXiv preprint arXiv:2210.04996*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Jermsak Jermsurawong and Nizar Habash. 2015. Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Chloe Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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*.

Yu-Wen Lo, Qiangfu Zhao, Yu-Hsien Ting, and Rung-Ching Chen. 2015. Automatic generation and recommendation of recipes based on outlier analysis. In *IEEE 7th International Conference on Awareness Science and Technology (iCAST)*.

Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. 2019. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):187–203.

Liangming Pan, Jingjing Chen, Jianlong Wu, Shaoteng Liu, Chong-Wah Ngo, Min-Yen Kan, Yugang Jiang, and Tat-Seng Chua. 2020. Multimodal cooking workflow construction for food recipes. In *ACM Multi-Media*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Simone Tedeschi, Valentino Maiorca, Niccolò Campolungo, Francesco Cecconi, and Roberto Navigli. 2021. WikiNEuRal: Combined neural and knowledge-based silver data creation for multilingual NER. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2521–2533, Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Kirstin Walter, Mirjam Minor, and Ralph Bergmann. 2011. Workflow extraction from cooking recipes. In *Proceedings of the ICCBR 2011 Workshops*.

Liping Wang, Qing Li, Na Li, Guozhu Dong, and Yu Yang. 2008. Substructure similarity measurement in chinese recipes. In *ACM WWW*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Qingrong Xia, Zhenghua Li, Rui Wang, and Min Zhang. 2021. Stacked amr parsing with silver data. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4729–4738.

Haoran Xie, Lijuan Yu, and Qing Li. 2010. A hybrid semantic item model for recipe search by example. In *IEEE International Symposium on Multimedia*.

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*.

Yoko Yamakata, Shinsuke Mori, and John A Carroll. 2020. English recipe flow graph corpus. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5187–5194.