Learning from Executions for Semantic Parsing

Bailin Wang, Mirella Lapata and **Ivan Titov**
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
`bailin.wang@ed.ac.uk, {mlap, ititov}@inf.ed.ac.uk`

Abstract

Semantic parsing aims at translating natural language (NL) utterances onto machine-interpretable programs, which can be executed against a real-world environment. The expensive annotation of utterance-program pairs has long been acknowledged as a major bottleneck for the deployment of contemporary neural models to real-life applications. In this work, we focus on the task of semi-supervised learning where a limited amount of annotated data is available together with many unlabeled NL utterances. Based on the observation that programs which correspond to NL utterances must be always executable, we propose to encourage a parser to generate executable programs for *unlabeled* utterances. Due to the large search space of executable programs, conventional methods that use approximations based on beam-search such as self-training and top-k marginal likelihood training, do not perform as well. Instead, we view the problem of learning from executions from the perspective of posterior regularization and propose a set of new training objectives. Experimental results on OVERNIGHT and GEOQUERY show that our new objectives outperform conventional methods, bridging the gap between semi-supervised and supervised learning.

1 Introduction

Semantic parsing is the task of mapping natural language (NL) utterances to meaning representations (aka programs) that can be executed against a real-world environment such as a knowledge base or a relational database. While neural sequence-to-sequence models (Dong and Lapata, 2016; Jia and Liang, 2016a) have achieved much success in this task in recent years, they usually require a large amount of labeled data (i.e., utterance-program pairs) for training. However, annotating utterances with programs is expensive as it requires expert knowledge of meaning representations (e.g., lambda calculus, SQLs) and the environment against which they are executed (e.g., a knowledge base, a relational database). An alternative to annotation is to collect answers (or denotations) of programs, rather than programs themselves (Liang et al., 2013; Berant et al., 2013). In this work, we focus on the more extreme setting where there are no annotations available for a large number of utterances. This setting resembles a common real-life scenario where massive numbers of user utterances can be collected when deploying a semantic parser (Iyer et al., 2017). Effectively utilizing the unlabeled data makes it possible for a semantic parser to improve over time without human involvement.

list all 3 star rated thai restaurants		
Program Candidates	Gold	Exe
select restaurant *where* star_rating = thai	✗	✗
select restaurant *where* cuisine > 3	✗	✗
select restaurant *where* star_rating = 3	✗	✓
select restaurant *where* star_rating = 3 *and* cuisine = thai	✓	✓

Figure 1: Candidate programs for an utterance can be classified by executability (Exe); note that the gold program is always in the set of executable programs. We propose to ultilize the weak yet freely available signal of executablility for learning.

Our key observation is that not all candidate programs for an utterance will be semantically valid. This implies that only some candidate programs can be executed and obtain non-empty execution results.[1] As illustrated in Figure 1, executability is a weak signal that can differentiate between semantically valid and invalid programs. On unlabeled utterances, we can encourage a parser to only focus on executable programs ignoring non-executable ones. Moreover, the executability of a program

[1]In the rest of this paper, we extend the meaning of 'executability', and use it to refer to the case where a program is executable and obtains non-empty results.

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2747–2759
June 6–11, 2021. ©2021 Association for Computational Linguistics

can be obtained from an executor for free without requiring human effort. Executability has previously been used to guide the decoding of a semantic parser (Wang et al., 2018). We take a step further to directly use this weak signal for learning from unlabeled utterances.

To learn from executability, we resort to marginal likelihood training, i.e., maximizing the marginal likelihood of all executable programs for an unlabeled NL utterance. However, the space of all possible programs is exponentially large, as well as the space of executable ones. Hence, simply marginalizing over all executable programs is intractable. Typical approximations use beam search to retrieve a handful of ('seen') programs, which are used to approximate the entire space. Using such approximations can lead to optimization getting trapped in undesirable local minima. For example, we observe that encouraging a model to exploit seen executable programs hinders exploration and reinforces the preference for shorter programs, as discussed in Section 6.3. This happens because shorter programs are both more likely to be among 'seen' programs (probably due to using locally-normalized autoregressive modeling) and more likely to be executable. To alleviate these issues, we derive three new alternative objectives, relying on a new interpretation of marginal likelihood training from the perspective of posterior regularization. Our proposed objectives encode two kinds of inductive biases: 1) **discouraging seen non-executable programs**, which plays a similar role to encouraging seen executable ones but does not share its drawback of hindering exploration; 2) **encouraging sparsity** among executable programs, which encourages a parser to only focus on a subset of executable programs by softly injecting a sparsity constraint. This is desirable, as there are only one or few correct programs for each utterance (see Figure 1), and an accurate parser should assign probability mass only to this subset. We collectively call these objectives X-PR, as a shorthand for Execution-guided Posterior Regularization.

We conduct experiments on two representative semantic parsing tasks: text-to-LF (logical form) parsing over a knowledge base and text-to-SQL (Zelle and Mooney, 1996) parsing over a relational database. Concretely, we evaluate our methods on the OVERNIGHT (Wang et al., 2015a) and GEOQUERY datasets. We simulate the semi-supervised learning setting by treating 70% of the training data as unlabeled. Empirical results show that our method can substantially boost the performance of a parser, trained only on labeled data, by utilizing a large amount of unlabeled data.

Our contributions are summarized as follows:

- We show how to exploit unlabeled utterances by taking advantage of their executability.
- To better learn from executability, we propose a set of new objectives based on posterior regularization.
- Our method can help a base parser achieve substantially better performance by utilizing unlabeled data.

Our code, datasets, and splits are publicly available at `https://github.com/berlino/tensor2struct-public`.

2 Related Work

Semi-Supervised Semantic Parsing In the context of semantic parsing, semi-supervised models using limited amounts of parallel data and large amounts of unlabeled data treat either utterances or programs as discrete latent variables and induce them in the framework of generative models (Kočiský et al., 2016; Yin et al., 2018). A challenge with these methods is that (combinatorially) complex discrete variables make optimization very hard, even with the help of variational inference. In this work, we seek to directly constrain the discriminative parser with signals obtained from executions. Our method can potentially be integrated into these generative models to regularize discrete variables.

(Underspecified) Sequence-Level Rewards There have been attempts in recent years to integrate sequence-level rewards into sequence-to-sequence training as a way of accommodating task-specific objectives. For example, BLEU can be optimized for coherent text generation (Bosselut et al., 2018) and machine translation (Wu et al., 2018) via reinforcement learning or beam-search (Wiseman and Rush, 2016). In this work, we resort to marginal likelihood training to exploit binary executability rewards for semantic parsing (i.e., whether a program is executable or not), which has been shown to be more effective than REINFORCE (Guu et al., 2017).

More importantly, our binary reward is underspecified, i.e., there exist many spurious programs that enjoy the same reward as the gold program.

This issue of learning from underspecified rewards underlies many weakly-supervised tasks, e.g., learning from denotations (Liang et al., 2013; Berant et al., 2013), weakly supervised question answering (Min et al., 2019). Previous work tried to model latent alignments (Wang et al., 2019) between NL and programs to alleviate this issue. In this work, we take an orthogonal direction and propose several training objectives that alleviate the impact of spurious programs.

Execution for Semantic Parsing Execution has been utilized in semantic parsing (Wang et al., 2018) and the related area of program synthesis (Chen et al., 2019). These approaches exploit the execution of partial programs to guide the search for plausible complete programs. Although partial execution is feasible for SQL-style programs, it cannot be trivially extended to general meaning representation (e.g., logical forms). In this work, we explore a more general setting where execution can be only obtained from complete programs.

3 Executability as Learning Signal

In this section, we formally define our semi-supervised learning setting and show how to incorporate executability into the training objective whilst relying on the marginal likelihood training framework. We also present two conventional approaches to optimizing marginal likelihood.

3.1 Problem Definition

Given a set of labeled NL-program pairs $\{(x_i^l, y_i^l)\}_{i=1}^N$ and a set of unlabeled NL utterances $\{x_j\}_{j=1}^M$, where N and M denote the sizes of the respective datasets, we would like to learn a neural parser $p(y|x, \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, that maps utterances to programs. The objective to minimize consists of two parts:

$$\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\text{sup}}(x_i^l, y_i^l) + \lambda \frac{1}{M}\sum_{j=1}^{M} \mathcal{L}_{\text{unsup}}(x_i) \tag{1}$$

where $\mathcal{L}_{sup}$ and $\mathcal{L}_{unsup}$ denote the supervised and unsupervised loss, respectively. For labeled data, we use the negative log-likelihood of gold programs; for unlabeled data, we instead use the log marginal likelihood (MML) of *all executable programs*. Specifically, they are defined as follows:

$$\mathcal{L}_{\text{sup}}(x, y) = -\log p(y|x, \boldsymbol{\theta}) \tag{2}$$

$$\mathcal{L}_{\text{unsup}}(x) = -\log \sum_{y} R(y) p(y|x, \boldsymbol{\theta}) \tag{3}$$

where $R(y)$ is a binary reward function that returns 1 if y is executable and 0 otherwise. In practice, this function is implemented by running a task-specific executor, e.g., a SQL executor.

Another alternative to unsupervised loss is REINFORCE (Sutton et al., 1999), i.e., maximize the expected $R(y)$ with respect to $p(y|x, \theta)$. However, as presented in Guu et al. (2017), this objective usually underperforms MML, which is consistent with our initial experiments.[2]

3.2 Self-Training and Top-K MML

MML in Equation (3) requires marginalizing over all executable programs which is intractable. Conventionally, we resort to beam search to explore the space of programs and collect executable ones. To illustrate, we can divide the space of programs into four parts based on whether they are executable and observed, as shown in Figure 2a. For example, programs in $P_{\text{SE}} \cup P_{\text{SN}}$ are seen in the sense that they are retrieved by beam search. Programs in $P_{\text{SE}} \cup P_{\text{UE}}$ are all executable, though only programs in P_{SE} can be directly observed.

Two common approximations of Equation (3) are Self-Training (ST) and Top-K MML, and they are defined as follows:

$$\mathcal{L}_{\text{ST}}(x, \boldsymbol{\theta}) = -\log p(y^*|x, \boldsymbol{\theta}) \tag{4}$$

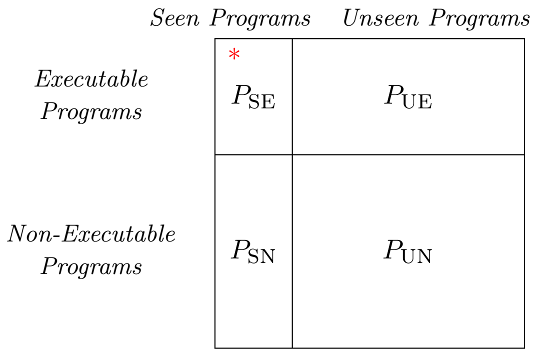
$$\mathcal{L}_{\text{top-k}}(x, \boldsymbol{\theta}) = -\log \sum_{y \in P_{\text{SE}}} p(y|x, \boldsymbol{\theta}) \tag{5}$$

where y^* denotes the most probable program, and it is approximated by the most probable one from beam search.

It is obvious that both methods only exploit programs in P_{SE}, i.e., executable programs retrieved by beam search. In cases where a parser successfully includes the correct programs in P_{SE}, both approximations should work reasonably well. However, if a parser is uncertain and P_{SE} does not contain the gold program, it would then mistakenly exploit incorrect programs in learning, which is problematic.

A naive solution to improve Self-Training or Top-K MML is to explore a larger space, e.g., increase the beam size to retrieve more executable

[2] We review the comparison between REINFORCE and MML in the appendix.



(a) Partitioned program space. Red asterisk denotes the most probable executable program y^*. P stands for program; subscript S stands for seen, U for unseen, E for executable, and N for non-executable.

$$\begin{aligned}
\mathcal{L}_{\text{ST}}(x,\boldsymbol{\theta}) &= -\log p(y^*|x,\boldsymbol{\theta}) \\
\mathcal{L}_{\text{top-k}}(x,\boldsymbol{\theta}) &= -\log \sum_{y\in P_{\text{SE}}} p(y|x,\boldsymbol{\theta}) \\
\mathcal{L}_{\text{repulsion}}(x,\boldsymbol{\theta}) &= -\log\big(1-\sum_{y\in P_{\text{SN}}} p(y|x,\boldsymbol{\theta})\big) \\
\mathcal{L}_{\text{gentle}}(x,\boldsymbol{\theta}) &= -p(P_{\text{SE}}\cup P_{\text{SN}})\log \sum_{y\in P_{\text{SE}}} p(y|x,\boldsymbol{\theta}) \\
&\quad - p(P_{\text{UE}}\cup P_{\text{UN}})\log \sum_{y\in P_{\text{UE}}\cup P_{\text{UN}}} p(y|x,\boldsymbol{\theta}) \\
\mathcal{L}_{\text{sparse}}(x,\boldsymbol{\theta}) &= -\sum_{y\in P_{\text{SE}}} q_{\text{sparse}}(y)\log p(y|x,\boldsymbol{\theta})
\end{aligned}$$

(b) Five objectives to approximate MML.

Figure 2: In (a) the program space is partitioned along two dimentions: executability and observability. In (b) we show two commonly used objectives (Self-Training and Top-K MML) and the three objectives proposed in this work.

programs. However, this would inevitably increase the computation cost of learning. We also show in the appendix that increasing beam size, after it exceeds a certain threshold, is no longer beneficial for learning. In this work, we instead propose better approximations without increasing beam size.

4 Method

We first present a view of MML in Equation (3) from the perspective of posterior regularization. This new perspective helps us derive three alternative approximations of MML: Repulsion MML, Gentle MML, and Sparse MML.

4.1 Posterior Regularization

Posterior regularization (PR) allows to inject linear constraints into posterior distributions of generative models, and it can be extended to discriminative models (Ganchev et al., 2010). In our case, we try to constrain the parser $p(y|x,\theta)$ to only assign probability mass to executable programs. Instead of imposing hard constraints, we softly penalize the parser if it is far away from a desired distribution $q(y)$, which is defined as $\mathbb{E}_q[R(y)] = 1$. Since R is a binary reward function, $q(y)$ is constrained to only place mass on executable programs whose rewards are 1. We denote all such desired distributions as the family $\mathcal{Q}$.

Specifically, the objective of PR is to penalize the KL-divergence between $\mathcal{Q}$ and p, which is:

$$\begin{aligned}
\mathcal{J}_{\mathcal{Q}}(\boldsymbol{\theta}) &= D_{\text{KL}}[\mathcal{Q}||p(y|x,\boldsymbol{\theta})] \\
&= \min_{q\in\mathcal{Q}} D_{\text{KL}}[q(y)||p(y|x,\boldsymbol{\theta})]
\end{aligned} \tag{6}$$

By definition, the objective has the following upper bound:

$$\begin{aligned}
\mathcal{J}(\boldsymbol{\theta},q) &= D_{\text{KL}}[q(y)||p(y|x,\boldsymbol{\theta})] \\
&= -\sum_{y} q(y)\log p(y|x,\boldsymbol{\theta}) - \mathcal{H}(q)
\end{aligned} \tag{7}$$

where $q \in \mathcal{Q}$, $\mathcal{H}$ denotes the entropy. We can use block-coordinate descent, an EM iterative algorithm to optimize it.

$$\begin{aligned}
\text{E}: q^{t+1} &= \arg\min_{q\in\mathcal{Q}} D_{\text{KL}}[q(y)||p(y|x,\boldsymbol{\theta}^t)] \\
\text{M}: \boldsymbol{\theta}^{t+1} &= \arg\min_{\boldsymbol{\theta}} -\sum_{y} q^{t+1}(y)[\log p(y|x,\boldsymbol{\theta})]
\end{aligned}$$

During the E-step, we try to find a distribution q from the constrained set $\mathcal{Q}$ that is closest to the current parser p in terms of KL-divergence. We then use q as a 'soft label' and minimize the cross-entropy between q and p during the M-step. Note that q is a constant vector and has no gradient wrt. $\boldsymbol{\theta}$ during the M-step.

The E-step has a closed-form solution:

$$q^{t+1}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{p(P_{\text{SE}}\cup P_{\text{UE}})} & y \in P_{\text{SE}}\cup P_{\text{UE}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $p(P_{\text{SE}} \cup P_{\text{UE}}) = \sum_{y' \in P_{\text{SE}} \cup P_{\text{UE}}} p(y'|x, \boldsymbol{\theta}^t)$. $q^{t+1}(y)$ is essentially a re-normalized version of p over executable programs. Interestingly, if we use the solution in the M-step, the gradient wrt. $\boldsymbol{\theta}$ is equivalent to the gradient of MML in Equation (3). That is, optimizing PR with the EM algorithm is equivalent to optimizing MML.[3] The connection between EM and MML is not new, and it has been well-studied for classification problems (Amini and Gallinari, 2002; Grandvalet and Bengio, 2004). In our problem, we additionally introduce PR to accommodate the executability constraint, and instantiate the general EM algorithm.

Although the E-step has a closed-form solution, computing q is still intractable due to the large search space of executable programs. However, this PR view provides new insight on what it means to approximate MML. In essence, conventional methods can be viewed as computing an approximate solution of q. Specifically, Self-Training corresponds to a delta distribution that only focuses on the most probable y^*.

$$q_{\text{ST}}^{t+1}(y) = \begin{cases} 1 & y = y^* \\ 0 & \text{otherwise} \end{cases}$$

Top-K MML corresponds to a re-normarlized distribution over P_{SE}.

$$q_{\text{top-k}}^{t+1}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{p(P_{\text{SE}})} & y \in P_{\text{SE}} \\ 0 & \text{otherwise} \end{cases}$$

Most importantly, this perspective leads us to deriving three new approximations of MML, which we collectively call X-PR.

4.2 Repulsion MML and Gentle MML

As mentioned previously, Self-Training and Top-K MML should be reasonable approximations in cases where gold programs are retrieved, i.e., they are in the seen executable subset (P_{SE} in Figure 2a). However, if a parser is uncertain, i.e., beam search cannot retrieve the gold programs, exclusively exploiting P_{SE} programs is undesirable. Hence, we consider ways of taking unseen executable programs (P_{UE} in Figure 2a) into account. Since we never directly observe unseen programs (P_{UE} or P_{UN}), our heuristics do not discriminate between executable and non-executable programs ($P_{\text{UE}} \cup P_{\text{UN}}$). In other words, upweighting P_{UE} programs will inevitably upweight P_{UN}.

[3]Please see the appendix for the proof and analysis of PR.

Based on the intuition that the correct program is included in either seen executable programs (P_{SE}) or unseen programs (P_{UE} and P_{UN}), we can simply push a parser away from seen non-executable programs (P_{SN}). Hence, we call such method Repulsion MML. Specifically, the first heuristic approximates Equation (8) as follows:

$$q_{\text{repulsion}}^{t+1}(y) = \begin{cases} \frac{p(y|x,\boldsymbol{\theta}^t)}{1-p(P_{\text{SN}})} & y \notin P_{\text{SN}} \\ 0 & \text{otherwise} \end{cases}$$

Another way to view this heuristic is that we distribute the probability mass from seen non-executable programs (P_{SN}) to other programs. In contrast, the second heuristic is more 'conservative' about unseen programs as it tends to trust seen executable P_{SN} programs more. Specifically, the second heuristic uses the following approximations to solve the E-step.

$$q_{\text{gentle}}^{t+1}(y) = \begin{cases} \frac{p(P_{\text{SE}\cup\text{SN}})}{p(P_{\text{SE}})} p(y|x, \boldsymbol{\theta}^t) & y \in P_{\text{SE}} \\ p(y|x, \boldsymbol{\theta}^t) & y \in P_{\text{UE}} \cup P_{\text{UN}} \\ 0 & y \in P_{\text{SN}} \end{cases}$$

Intuitively, it shifts the probability mass of seen non-executable programs (P_{SN}) directly to seen executable programs (P_{SE}). Meanwhile, it neither upweights nor downweights unseen programs. We call this heuristic Gentle MML. Compared with Self-Training and Top-K MML, Repulsion MML and Gentle MML lead to better exploration of the program space, as only seen non-executable (P_{SN}) programs are discouraged.

4.3 Sparse MML

Sparse MML is based on the intuition that in most cases there is only one or few correct programs among all executable programs. As mentioned in Section 2, spurious programs that are executable, but do not reflect the semantics of an utterance are harmful. One empirical evidence from previous work (Min et al., 2019) is that Self-Training outperforms Top-K MML for weakly-supervised question answering. Hence, exploiting all seen executable programs can be sub-optimal. Following recent work on sparse distributions (Martins and Astudillo, 2016; Niculae et al., 2018), we propose to encourage sparsity of the 'soft label' q. Encouraging sparsity is also related to the minimum entropy and low-density separation principles which are commonly used in semi-supervised learning (Grandvalet and Bengio, 2004; Chapelle and Zien, 2005).

To achieve this, we first interpret the entropy term $\mathcal{H}$ in Equation (7) as a regularization of q. It is known that entropy regularization always results in a dense q, i.e., all executable programs are assigned non-zero probability. Inspired by SparseMax (Martins and Astudillo, 2016), we instead use L2 norm for regularization. Specifically, we replace our PR objective in Equation (7) with the following one:

$$\mathcal{J}_{\text{sparse}}(\boldsymbol{\theta}, q) = -\sum_{y} q(y) \log p(y|s, \boldsymbol{\theta}) + \frac{1}{2}||q||_2^2$$

where $q \in \mathcal{Q}$. Similarly, it can be optimized by the EM algorithm:

$$\begin{aligned} \text{E}: q^{t+1} &= \text{SparseMax}_{\mathcal{Q}}(\log p(y|x, \boldsymbol{\theta}^t)) \\ \text{M}: \boldsymbol{\theta}^{t+1} &= \arg\min_{\boldsymbol{\theta}} -\sum_{y} q^{t+1}(y)[\log p(y|x, \boldsymbol{\theta})] \end{aligned}$$

where the top-E-step can be solved by the SparseMax operator, which denotes the Euclidean projection from the vector of logits $\log p(y|x, \boldsymbol{\theta}^t)$ to the simplex $\mathcal{Q}$. Again, we solve the E-step approximately. One of the approximations is to use top-k SparseMax which constrain the number of non-zeros of q to be less than k. It can be solved by using a top-k operator and followed by SparseMax (Correia et al., 2020). In our case, we use beam search to approximate the top-k operator and the resulting approximation for the E-step is defined as follows:

$$q_{\text{sparse}}^{t+1} = \text{SparseMax}_{y \in P_{\text{SE}}} \left(\log p(y|x, \boldsymbol{\theta}^t) \right)$$

Intuitively, q_{sparse}^{t+1} occupies the middle ground between Self-Training (uses y^* only) and Top-K MML (uses all P_{SE} programs). With the help of sparsity of q introduced by SparseMax, the M-step will only promote a subset of P_{SE} programs.

Summary We propose three new approximations of MML for learning from executions. They are designed to complement Self-Training and Top-K MML via discouraging seen non-executable programs and introducing sparsity. In the following sections, we will empirically show that they are superior to Self-Training and Top-K MML for semi-supervised semantic parsing. The approximations we proposed may also be beneficial for learning from denotations (Liang et al., 2013; Berant et al., 2013) and weakly supervised question answering (Min et al., 2019), but we leave this to future work.

5 Semantic Parsers

In principle, our X-PR framework is model-agnostic, i.e., it can be coupled with any semantic parser for semi-supervised learning. In this work, we use a neural parser that achieves state-of-the-art performance across semantic parsing tasks. Specifically, we use RAT-SQL (Wang et al., 2020) which features a relation-aware encoder and a grammar-based decoder. The parser was originally developed for text-to-SQL parsing, and we adapt it to text-to-LF parsing. In this section, we briefly review the encoder and decoder of this parser. For more details, please refer to Wang et al. (2020).

5.1 Relation-Aware Encoding

Relation-aware encoding is originally designed to handle *schema encoding* and *schema linking* for text-to-SQL parsing. We generalize these two notions for both text-to-LF and text-to-SQL parsing as follows:

- *enviroment encoding*: encoding enviroments, i.e., a knowledge base consisting of a set of triples; a relational database represented by its schema
- *enviroment linking*: linking mentions to intended elements of environments, i.e., mentions of entities and properties of knowledge bases; mentions of tables and columns of relational databases

Relation-aware attention is introduced to inject discrete relations between environment items, and between the utterance and environments into the self-attention mechanism of Transformer (Devlin et al., 2019). The details of relation-aware encoding can be found in the appendix.

5.2 Grammar-Based Decoding

Typical sequence-to-sequence models (Dong and Lapata, 2016; Jia and Liang, 2016a) treat programs as sequences, ignoring their internal structure. As a result, the well-formedness of generated programs cannot be guaranteed. Grammar-based decoders aim to remedy this issue. For text-to-LF parsing, we use the type-constrained decoder proposed by Krishnamurthy et al. (2017); for text-to-SQL parsing, we use an AST (abstract syntax tree) based decoder following Yin and Neubig (2018). Note that grammar-based decoding can only ensure the syntactic correctness of generated programs. Executable programs are additionally semantically correct. For example, all programs in Figure 1 are

Model	OVERNIGHT									GEO
	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	Avg.	
Lower bound	82.2	54.1	64.9	61.4	64.3	72.7	71.7	76.7	68.5	60.6
Self-Traing	84.7	52.6	67.9	59.8	68.6	80.1	71.1	77.4	70.3	64.2
Top-K MML	83.1	55.3	68.5	56.9	67.7	73.7	69.9	76.2	68.9	61.3
Repulsion MML	**84.9**	56.3	70.8	60.9	70.3	79.8	72.0	**78.3**	71.7	64.9
Gentle MML	84.1	58.1	70.2	**63.0**	71.5	78.7	72.3	76.4	71.8	65.6
Sparse MML	83.9	**58.6**	**72.6**	60.3	**75.2**	**80.6**	**72.6**	77.8	**72.7**	**67.4**
Upper Bound	87.7	62.9	82.1	71.4	78.9	82.4	82.8	80.8	78.6	74.2

Table 1: Execution accuracy of supervised and semi-supervised models on all domains of OVERNIGHT and GEOQUERY. In semi-supervised learning, 30% of the original training examples are treated as labeled and the remaining 70% as unlabeled. Lower bound refers to supervised models that only use labeled examples and discard unlabeled ones whereas upper bound refers to supervised models that have access to gold programs of unlabeled examples. Avg. refers to the average accuracy of the eight OVERNIGHT domains. We average runs over three random splits of the original training data for semi-supervised learning.

well-formed, but the first two programs are semantically incorrect.

6 Experiments

To evaluate X-PR, we present experiments on semi-supervised semantic parsing. We also analyze how the objectives affect the training process.

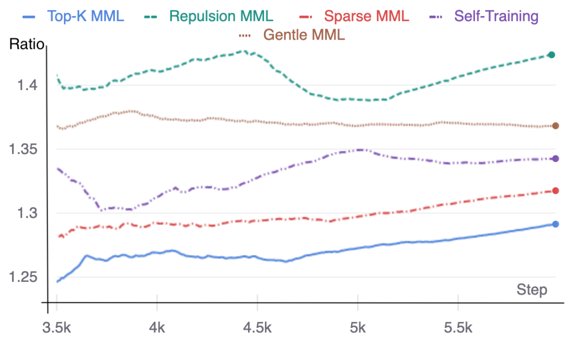
6.1 Semi-Supervised Learning

We simulate the setting of semi-supervised learning on standard text-to-LF and text-to-SQL parsing benchmarks. Specifically, we randomly sample 30% of the original training data as the labeled data, and use the rest 70% as the unlabeled data. For text-to-LF parsing, we use the OVERNIGHT dataset (Wang et al., 2015a), which has eight different domains, each with a different size ranging between 801 and 4,419; for text-to-SQL parsing, we use GEOQUERY (Zelle and Mooney, 1996) which contains 880 utterance-SQL pairs. The semi-supervised setting is very challenging as leveraging only 30% of the original training data would result in only around 300 labeled examples in four domains of OVERNIGHT and also in GEOQUERY.

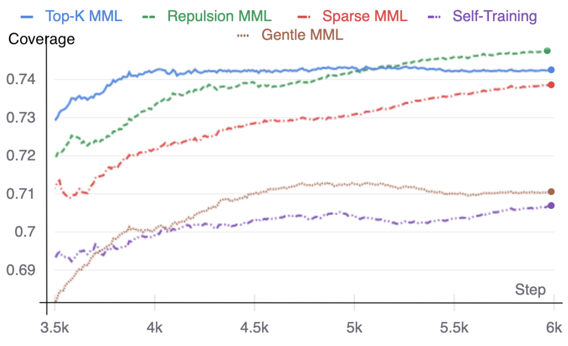
Supervised Lower and Upper Bounds As baselines, we train two supervised models. The first one only uses the labeled data (30% of the original training data) and discards the unlabeled data in the semi-supervised setting. We view this baseline as a *lower bound* in the sense that any semi-supervised method is expected to surpass this. The second one has extra access to gold programs for the unlabeled data in the semi-supervised setting, which means it uses the full original training data. We view this baseline as an *upper bound* for semi-supervised learning; we cannot expect to approach it as the executability signal is much weaker than direct supervision. By comparing the performance of the second baseline (upper bound) with previous methods (Jia and Liang, 2016b; Herzig and Berant, 2017; Su and Yan, 2017), we can verify that our semantic parsers are state-of-the-art. Please refer to the Appendix for detailed comparisons. Our main experiments aim to show how the proposed objectives can mitigate the gap between the lower- and upper-bound baselines by utilizing 70% unlabeled data.

Semi-Supervised Training and Tuning We use stochastic gradient descent to optimize Equation (1). At each training step, we sample two batches from the labeled and unlabeled data, respectively. In preliminary experiments, we found that it is crucial to pre-train a parser on supervised data alone; this is not surprising as all of the objectives for learning from execution rely on beam search which would only introduce noise with an untrained parser. That is, λ in Equation (1) is set to 0 during initial updates, and is switched to a normal value afterwards.

We leave out 100 labeled examples for tuning the hyperparameters. The hyperparameters of the semantic parser are only tuned for the development of the supervised baselines, and are fixed for semi-supervised learning. The only hyperparameter we tune in the semi-supervised setting is the λ in Equation (1), which controls how much the unsupervised objective influences learning. After tuning, we use all the labeled examples for supervised training and use the last checkpoints for evaluation on the test set.



(a) Average Ratios.



(b) Coverage of gold programs.

Figure 3: Effect of different learning objectives in terms of average ratios and coverage (view in color).

6.2 Main Results

Our experiments evaluate the objectives presented in Figure 2 under a semi-supervised learning setting. Our results are shown in Table 1.

Self-Training and Top-K MML First, Top-K MML, which exploits more executable programs than Self-Training, does not yield better performance in six domains of OVERNIGHT and GEOQUERY. This observation is consistent with Min et al. (2019) where Top-K MML underperforms Self-Training for weakly-supervised question answering. Self-Training outperforms the lower bound in five domains of OVERNIGHT, and on average. In contrast, Top-K MML obtains a similar performance to the lower bound in terms of average accuracy.

X-PR Objectives In each domain of OVERNIGHT and GEOQUERY, the objective that achieves the best performance is always within X-PR. In terms of average accuracy in OVERNIGHT, all our objectives perform better than Self-Training and Top-K MML. Among X-PR, Sparse MML performs best in five domains of OVERNIGHT, leading to a margin of 4.2% compared with the lower bound in terms of average accuracy. In GEOQUERY, Sparse MML also obtain best performance.

Based on the same intuition of discouraging seen non-executable programs, Repulsion MML achieves a similar average accuracy to Gentle MML in OVERNIGHT. In contrast, Gentle MML tends to perform better in domains whose parsers are weak (such as HOUSING, BLOCKS) indicated by their lower bounds. In GEOQUERY, Gentle MML performs slightly better than Repulsion MML. Although it does not perform better than Repulsion MML, it retrieves more accurate programs and also generates longer programs (see next section for details).

To see how much labeled data would be needed for a supervised model to reach the same accuracy as our semi-supervised models, we conduct experiments using 40% of the original training examples as the labeled data. The supervised model achieves 72.6% on average in OVERNIGHT, implying that 'labeling' 33.3% more examples would yield the same accuracy as our best-performing objective (Sparse MML).

6.3 Analysis

To better understand the effect of different objectives, we conduct analysis on the training process of semi-supervised learning. For the sake of brevity, we focus our analysis on the CALENDAR domain but have drawn similar conclusions for the other domains.

Length Ratio During preliminary experiments, we found that all training objectives tend to favor short executable programs for unlabeled utterances. To quantify this, we define the metric of average ratio as follows:

$$ratio = \frac{\sum_i \sum_{y \in P_{\text{SE}}(x_i)} |y|}{\sum_i |x_i||P_{\text{SE}}(x_i)|} \qquad (9)$$

where $P_{\text{SE}}(x_i)$ denotes seen executable programs of x_i, $|x|, |y|$ denotes the length of an utterance and a program, respectively, and $|P_{\text{SE}}(x_i)|$ denotes the number of seen executable programs. Intuitively, average ratio reveals the range of programs that an objective is exploiting in terms of length. This metric is computed in an online manner, and x_i is a sequence of data fed to the training process.

As shown in Figure 3a, Top-K MML favors shorter programs, especially during the initial steps. In contrast, Repulsion MML and Gentle MML prefer longer programs. For reference, we can compute the gold ratio by assuming $P_{\text{SE}}(x_i)$ only contains the gold program. The gold ratio for CALENDAR is 2.01, indicating that all objectives are still preferring programs that are shorter than gold programs. However, by not directly exploiting seen executable programs, Repulsion MML and Gentle MML alleviate this issue compared with Top-K MML.

Coverage Next, we analyze how much an objective can help a parser retrieve gold programs for unlabeled data. Since the orignal data contains the gold programs for the unlabeled data, we ultilize them to define the metric of coverage as follows:

$$coverage = \frac{\sum_i I[\hat{y}_i \in P_{\text{SE}}(x_i)]}{\sum_i |x_i|} \tag{10}$$

where I is an indicator function, $\hat{y}_i$ denotes the gold program of an utterance x_i. Intuitively, this metric measures how often a gold program is captured in P_{SE}. As shown in Figure 3b, Self-Training, which only exploits one program at a time, is relatively weak in terms of retrieving more gold programs. In contrast, Repulsion MML retrieves more gold programs than the others.

As mentioned in Section 4.3, SparseMax can be viewed as an interpolation between Self-Training and Top-K MML. This is also reflected in both metrics: Sparse MML always occupies the middle-ground performance between ST and Top-K MML. Interestingly, although Sparse MML is not best in terms of both diagnostic metrics, it still achieves the best accuracy in this domain.

7 Conclusion

In this work, we propose to learn a semi-supervised semantic parser from the weak yet freely available executability signals. Due to the large search space of executable programs, conventional approximations of MML training, i.e, Self-Training and Top-K MML, are often sub-optimal. We propose a set of alternative objectives, namely X-PR, through the lens of posterior regularization. Empirical results on semi-supervised learning show that X-PR can help a parser achieve substantially better performance than conventional methods, further bridging the gap between semi-supervised learning and supervised learning. In the future, we would like to extend X-PR to related tasks such as learning from denotations and weakly supervised question answering.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments. We gratefully acknowledge the support of the European Research Council (Titov: ERC StG BroadSem 678254; Lapata: ERC CoG TransModal 681760) and the Dutch National Science Foundation (NWO VIDI 639.022.518).

References

Massih-Reza Amini and Patrick Gallinari. 2002. Semi-supervised logistic regression. In *ECAI*, pages 390–394.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Antoine Bosselut, Asli Celikyilmaz, Xiaodong He, Jianfeng Gao, Po-Sen Huang, and Yejin Choi. 2018. Discourse-aware neural rewards for coherent text generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 173–184, New Orleans, Louisiana. Association for Computational Linguistics.

Olivier Chapelle and Alexander Zien. 2005. Semi-supervised classification by low density separation. In *AISTATS*, volume 2005, pages 57–64. Citeseer.

Xinyun Chen, Chang Liu, and Dawn Song. 2019. Execution-guided neural program synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Gonçalo M. Correia, Vlad Niculae, Wilker Aziz, and André F. T. Martins. 2020. Efficient marginalization of discrete and structured latent variables via sparsity. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. 2010. Posterior regularization for structured latent variable models. *The Journal of Machine Learning Research*, 11:2001–2049.

Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 529–536.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2017. Neural Semantic Parsing over Multiple Knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 623–628, Stroudsburg, PA, USA.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016a. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016b. Data Recombination for Neural Semantic Parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Stroudsburg, PA, USA.

Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087, Austin, Texas. Association for Computational Linguistics.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Percy Liang, Michael I. Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

André F. T. Martins and Ramón Fernandez Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1614–1623. JMLR.org.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard EM approach for weakly supervised question answering. 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 2851–2864, Hong Kong, China. Association for Computational Linguistics.

Vlad Niculae, André F. T. Martins, Mathieu Blondel, and Claire Cardie. 2018. Sparsemap: Differentiable sparse structured inference. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3796–3805. PMLR.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246, Copenhagen, Denmark.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning semantic parsers from denotations with latent structured alignments and abstract programs. 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 3774–3785, Hong Kong, China. Association for Computational Linguistics.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015a. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015b. Building a Semantic Parser Overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1332–1342, Stroudsburg, PA, USA.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. A study of reinforcement learning for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3612–3621, Brussels, Belgium. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. 2018. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 754–765, Melbourne, Australia. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

A Method

MML vs. RL Another choice for learning from executions is to maximize the expected reward:

$$\mathcal{J}_{\text{RL}}(x, \boldsymbol{\theta}) = \mathbb{E}_{p(y|x,\boldsymbol{\theta})}[R(y)] \tag{11}$$

Recall that our MML objective is defined as:

$$\mathcal{J}_{\text{MML}}(x, \boldsymbol{\theta}) = \log \sum_{y} R(y) p(y|x, \boldsymbol{\theta}) \tag{12}$$

As shown in previous work (Guu et al., 2017), the gradients (wrt. $\boldsymbol{\theta}$) of RL and MML have the same form:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(x, \boldsymbol{\theta}) = \sum_{y} q(y) \nabla_{\boldsymbol{\theta}} \log p(y|x, \boldsymbol{\theta}) \tag{13}$$

where q can be viewed as a soft-label that is induced from p. For RL, $q_{\text{RL}}(y) = p_{\boldsymbol{\theta}}(y|x, \theta)R(y)$; for MML, $q_{\text{MML}}(y) = \frac{R(y)p_{\boldsymbol{\theta}}(y|x)}{\sum_{y'} R(y')p_{\boldsymbol{\theta}}(y'|x)} = p_{\boldsymbol{\theta}}(y|x, R(y) = 1)$. Compared with RL, MML additionally renormalize the p over executable programs to obtain q. As a result, MML outputs a 'stronger' gradient than RL due to the renormalization. In our initial experiments, we found that RL is hard to converge whereas MML is not.

Proof of the E-Step Solution Denote the set of executable programs as $\mathcal{V}$. Since $q(y)$ is 0 for non-executable programs, we only need to compute it for executable programs in $\mathcal{V}$. We need to solve the following optimization problem:

$$\begin{aligned} \min_{q} & - \sum_{y \in \mathcal{V}} q(y) \log p(y|x, \boldsymbol{\theta}) + \sum_{y \in \mathcal{V}} q(y) \log q(y) \\ s.t. & \quad \sum_{y \in \mathcal{V}} q(y) = 1 \\ & \quad q(y) \geq 0 \end{aligned} \tag{14}$$

By solving it with KKT conditions, we can see that $q(y) \propto p(y|x, \boldsymbol{\theta})$. Since $q(y)$ needs to sum up to 1, it is easy to obtain that $q(y) = \frac{p(y|x,\boldsymbol{\theta})}{\sum_{y \in \mathcal{V}} p(y|x,\boldsymbol{\theta})}$.

B Experiments

Relation-Aware Encoding of Semantic Parsers Relation-aware encoding was originally introduced for text-to-SQL parsing in Wang et al. (2020) to accommodate discrete relations among schema items (e.g., tables and columns) and linking between an

Type of x	Type of y	Edge label	Description
Entity	Entity	RELATED-F RELATED-R	there exists a property p s.t. $(x, p, y) \in \mathcal{K}$ there exists a property p s.t. $(y, p, x) \in \mathcal{K}$
Entity	Property	HAS-PROPERTY-F HAS-PROPERTY-R	there exists an entity e s.t. $(x, y, e) \in \mathcal{K}$ there exists an entity e s.t. $(e, y, x) \in \mathcal{K}$
Property	Entity	PROP-TO-ENT-F PROP-TO-ENT-R	there exists an entity e s.t. $(y, x, e) \in \mathcal{K}$ there exists an entity e s.t. $(e, x, y) \in \mathcal{K}$
Utterance Token	Entity	EXACT-MATCH PARTIAL-MATCH	x and y are the same word token x is contained in entity y
Entity	Utterance Token	EXACT-MATCH-R PARTIAL-MATCH-R	y and x are the same word token y is contained in entity x
Utterance Token	Property	P-EXACT-MATCH P-PARTIAL-MATCH	x and y are the same word token x is contained in property y
Property	Utterance Token	P-EXACT-MATCH-R P-PARTIAL-MATCH-R	y and x are the same word token y is contained in property y

Table 2: Relation types used for text-to-LF parsing.

utterance and schema items. Let $\{\boldsymbol{x}_i^{(0)}\}_{i=0}^{N-1}$ denote the input to the parser consisting of NL tokens and the (linearized version of) environment; relation-aware encoding changes the multi-head self-attention (with H heads and hidden size d_x) as follows:

$$\begin{aligned} e_{ij}^{(n,h)} &= \frac{\boldsymbol{x}_i^{(n)} W_Q^{(n,h)} (\boldsymbol{x}_j^{(n)} W_K^{(n,h)} + \boldsymbol{r}_{ij}^K)^\top}{\sqrt{d_z/H}} \\ \alpha_{ij}^{(n,h)} &= \text{softmax}_j \{ e_{ij}^{(n,h)} \} \\ \boldsymbol{z}_i^{(n,h)} &= \sum_{j=1}^{n} \alpha_{ij}^{(n,h)} (\boldsymbol{x}_j^{(n)} W_V^{(n,h)} + \boldsymbol{r}_{ij}^V). \end{aligned} \tag{15}$$

where $\alpha_{ij}^{(n,h)}$ denotes the attention weights of head h at layer n, $0 \le h < H, 0 \le n < N$, and $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in \mathbb{R}^{d_x \times (d_x/H)}$. Most importantly, $\boldsymbol{r}_{ij}^K$ and $\boldsymbol{r}_{ij}^V$ are key and value embeddings of the discrete relation r_{ij} between items i and j. They are incorporated to bias attention towards discrete relations.

In this work, we re-use the relations from Wang et al. (2020) for our text-to-SQL parsing task on the GEOQUERY dataset. For text-to-LF parsing on the OVERNIGHT dataset, we elaborate on how we define discrete relations. The input of text-to-LF parsing is an utterance and a fixed knowledge base $\mathcal{K}$ which is represented as a set of triples in the form of (entity1, property, entity2). To feed the input into a transformer, we first linearize it into an ordered sequence which contains a list of utterance tokens, followed by a sequence of entities and properties. To model the relations among the items of this sequence, we define relations between each pair of items, denoted by (x, y), in Table 2.

Statitics of Datasets The numbers of examples in each domain are shown in Table 3. Four domains of OVERNIGHT and GEOQUERY contain only around 1000 examples.

Results of Supervised Models The results of supervised models (upper bounds) are shown in Table 4. Our parser achieves the best performance among models without cross-domain training. This confirms that we use a strong base parser for our experiments on semi-supervised learning.

Beam Size We try the beam size from $\{4, 8, 16, 32, 64\}$ and finally picks 16 which performs best in the OVERNIGHT dataset. We also try a larger beam size of 128 during preliminary experiments. However, the model is extremely slow to train and does not outperform the one with beam size 16.

Semi-Supervised Learning with 10% Data We also investigate a semi-supervised setting where only 10% of the original training data are treated as labeled data. This is more challenging as four domains of OVERNIGHT and GEOQUERY only have around 100 labeled utterance-program pairs in this setting. The results are shown in Table 5. In terms of average accuracy, Sparse MML achieves the best performance. The margin of improvement, compared to the lower bound, is relatively smaller than using 30% data (3% vs 4.2%). As all objectives rely on beam search, a weak base parser, as indicated by the lower bound, is probably

	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	GEO
all	1952	1995	837	941	801	1080	1657	4419	880

Table 3: Data sizes of the OVERNIGHT and GEOQUERY dataset.

Model	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	Avg.
Wang et al. (2015b)	46.3	41.9	74.4	54.5	59.0	70.8	75.9	48.2	58.8
Jia and Liang (2016b)	85.2	58.1	78.0	71.4	76.4	79.6	76.2	81.4	75.8
Herzig and Berant (2017)	85.2	61.2	77.4	67.7	74.5	79.2	79.5	80.2	75.6
Su and Yan (2017)	86.2	60.2	79.8	71.4	78.9	84.7	81.6	82.9	78.2
Ours	87.7	62.9	82.1	71.4	78.9	82.4	82.8	80.8	**78.6**
Herzig and Berant (2017)*	86.2	62.7	82.1	78.3	80.7	82.9	82.2	81.7	79.6
Su and Yan (2017)*	88.2	62.7	82.7	78.8	80.7	86.1	83.7	83.1	80.8

Table 4: Test accuracy of supervised models on all domains for OVERNIGHT. Models with * are augmented with cross-domain training.

	OVERNIGHT									GEO
Model	BASKETBALL	BLOCKS	CALENDAR	HOUSING	PUBLICATIONS	RECIPES	RESTAURANTS	SOCIAL	Avg.	
Lower Bound	68.0	35.3	40.5	34.4	39.1	45.8	59.3	63.7	48.3	42.7
Self-Traing	66.5	37.8	**48.2**	**39.2**	41.0	41.2	**63.0**	63.8	50.1	44.1
Top-K MML	70.3	36.8	42.3	30.7	37.3	44.4	61.1	62.2	48.1	40.1
Repulsion MML	70.8	37.3	44.6	37.8	39.8	44.4	60.2	**67.8**	50.3	44.8
Gentle MML	68.3	38.1	45.8	**39.2**	**43.5**	**44.9**	59.9	64.0	50.5	**46.6**
Sparse MML	**73.4**	**42.1**	46.4	38.1	42.2	43.1	**63.0**	64.9	**51.3**	45.2
Upper Bound	87.7	62.9	82.1	71.4	78.9	82.4	82.8	80.8	78.6	74.2

Table 5: Execution accuracy of supervised and semi-supervised models on all domains of OVERNIGHT and GEOQUERY. In semi-supervised learning, 10% of the original training examples are treated as labeled and the remaining 90% as unlabeled. Lower bound refers to supervised models that only use labeled examples and discard unlabeled ones whereas upper bound refers to supervised models that have access to gold programs of unlabeled examples. Avg. refers to the average accuracy of the eight OVERNIGHT domains.

harder to improve. By taking account of unseen programs, Repulsion MML and Gentle MML tend to be more useful in domains such as HOUSING, PUBLICATIONS, RECIPES where the base parsers are very weak (accuracy is around 40%). Moreover, Gentle MML performs best in GEOQUERY.