Code Generation from Natural Language with Less Prior and More Monolingual Data

Sajad Norouzi[*]   **Keyi Tang**   **Yanshuai Cao**

Borealis AI

sajadn@cs.toronto.edu,{keyi.tang,yanshuai.cao}@borealisai.com

Abstract

Training datasets for semantic parsing are typically small due to the higher expertise required for annotation than most other NLP tasks. As a result, models for this application usually need additional prior knowledge to be built into the architecture or algorithm. The increased dependency on human experts hinders automation and raises the development and maintenance costs in practice. This work investigates whether a generic transformer-based seq2seq model can achieve competitive performance with minimal code-generation-specific inductive bias design. By exploiting a relatively sizeable monolingual corpus of the target programming language, which is cheap to mine from the web, we achieved 81.03% exact match accuracy on Django and 32.57 BLEU score on CoNaLa. Both are SOTA to the best of our knowledge. This positive evidence highlights a potentially easier path toward building accurate semantic parsers in practice. [†]

1 Introduction

For a machine to act upon users' natural language inputs, a model needs to convert the natural language utterances to machine-understandable meaning representation, i.e. semantic parsing (SP). The output meaning representation is beyond shallow identification of topic, intention, entity or relation, but complex structured objects expressed as logical forms, query language or general-purpose programs. Therefore, annotating parallel corpus for semantic parsing requires more costly expertise.

SP shares some resemblance with machine translation (MT). However, SP datasets are typically smaller, with only a few thousand to at most tens of thousands of examples, even smaller than most low resource MT problems. Simultaneously, because
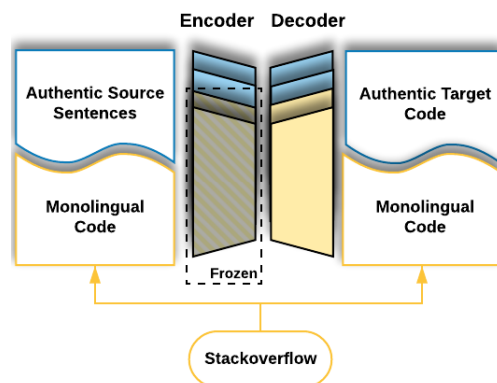

Figure 1: TAE: the monolingual corpus is used both as source and target. The encoder is frozen in the computation branch on the monolingual data.

the predicted outputs generally need to be exactly correct to execute and produce the right answer, the accuracy requirement is generally higher than MT. As a result, inductive bias design in architecture and algorithm has been prevalent in the SP literature (Dong and Lapata, 2016; Yin and Neubig, 2017, 2018; Dong and Lapata, 2018; Guo et al., 2019; Wang et al., 2019; Yin and Neubig, 2019).

While their progress is remarkable, excessive task-specific expert design makes the models complicated, hard to transfer to new domains, and challenging to deploy in real-world applications. In this work, we look at the opposite end of the spectrum and try to answer the following question: *with little inductive bias in the model, and no additional labelled data, is it still possible to achieve competitive performance?* This is an important question, as the answer could point to a much shorter road to practical SP without breaking the bank.

This paper shows that the answer is encouragingly affirmative. By exploiting a relatively large monolingual corpus of the programming language, a transformer-based Seq2Seq model (Vaswani et al., 2017) with little SP specific prior could potentially attain results superior to or competitive with the

---

[†]Code at https://github.com/BorealisAI/code-gen-TAE

state-of-the-art models specially designed for semantic parsing. Our contributions are three-fold:

- We provide evidence that transformer-based seq2seq models can reach a competitive or superior performance with models specifically designed for semantic parsing. This suggests an alternative route for future progress other than inductive bias design;

- We do empirical analysis over previously proposed approaches for incorporating monolingual data and show the effectiveness of our modified technique on a range of datasets;

- We set the new state-of-the-art on Django (Oda et al., 2015) reaching 81.03% exact match accuracy and on CoNaLa (Yin et al., 2018) with a BLEU score of 32.57.

2 Previous Work on Semantic Parsing

Different sources of prior knowledge about the SP problem structure could be exploited.

Input structure: Wang et al. (2019) adapts the transformer relative position encoding (Shaw et al., 2018) to express relations among the database schema elements as well as with the input text spans. Herzig and Berant (2020) proposed a span-based neural parser with compositional inductive bias built-in. Herzig and Berant (2020) also leverages a CKY-style (Cocke, 1969; Kasami, 1966; Younger, 1967) inference to link input features to output codes.

Output structure: ==The implicit tree or graph-like structures in the programs can also be exploited.== Dong and Lapata (2016) proposed parent-feeding LSTM following the tree structure. Dong and Lapata (2018) proposed a coarse-to-fine decoding approach. Guo et al. (2019) crafted an intermediate meaning representation to bridge the large gap between input utterance and the output SQL queries. Yin and Neubig (2017, 2018) proposed TranX, a more general-purpose transition-based system, to ensure grammaticality of predictions. Using TranX, the neural model predicts the linear sequence of AST-tree constructing actions instead of the program tokens. However, a human expert needs to craft the grammar, and the design quality impacts the learning and generalization for the neural nets.

Sequential models with less SP specific priors have been investigated (Dong and Lapata, 2016; Ling et al., 2016b; Zeng et al., 2020), However,

they generally fell short in accuracy comparing to the best of structure-exploiting ones listed above.

The most closely related to ours is the work by Xu et al. (2020) for incorporating external knowledge from extra datasets, which used a noisy parallel dataset from Stackoverflow to pre-train the SP and fine-tuned it on the primary dataset. Their approach's main limitation is still the need for (noisy) parallel data, albeit cheaper than the primary labelled set. Nonetheless, as we shall see in the experiment section later, our approach achieves better results when using the same amount of data mined from the same source despite ignoring the source sentence.

3 Background and Methodology

BERT (Devlin et al., 2018) class of pre-trained models can make up for the lack of inductive bias on the input side to some degree. On the output side, we hope to learn the necessary prior knowledge about the target meaning representation from unlabelled monolingual data.

Using monolingual data to improve seq2seq models is not new and has been extensively studied in MT before. Notable methods include *fusion* (Gulcehre et al., 2015; Ramachandran et al., 2016; Sriram et al., 2018; Stahlberg et al., 2018), *back-translation (BT)* (Sennrich et al., 2015; Edunov et al., 2018; Hoang et al., 2018), (Currey et al., 2017; Burlot and Yvon, 2018, 2019), and *BT with copied monolingual data* (Currey et al., 2017; Burlot and Yvon, 2019). However, due to more structured outputs, less training data, and different evaluation metrics of exact match correctness instead of BLEU, it is unclear if these lessons transfer from MT to SP. So SP-specific investigation is needed.

3.1 Target Autoencoding with Frozen Encoder

We assume a parallel corpus of natural language utterances and their corresponding programs, $\mathcal{B} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}$. The goal is to train a translator model (TM) to maximize the conditional log probability of $\boldsymbol{y}_i$ given $\boldsymbol{x}_i$, $T_{\boldsymbol{\theta}}(\boldsymbol{y}_i|\boldsymbol{x}_i)$, over the training set: $\mathcal{L}_{\text{sup}} = \sum_{\mathcal{B}} T_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$ where $\boldsymbol{\theta}$ is the vector of TM model parameters. Let $\mathcal{M} = \{\boldsymbol{y}_i'\}$ denote the ==monolingual dataset== in the target language.

Currey et al. (2017); Burlot and Yvon (2019) demonstrated that in low resource MT, auto-encoding the monolingual data besides the main supervised training is helpful. Following the same path, we add an auto-encoding objective term on monolingual data: $\mathcal{L}_{\text{full}} = \mathcal{L}_{\text{sup}} + \sum_{\mathcal{M}} T_{\boldsymbol{\theta}}(\boldsymbol{y}_i' \mid \boldsymbol{y}_i')$.
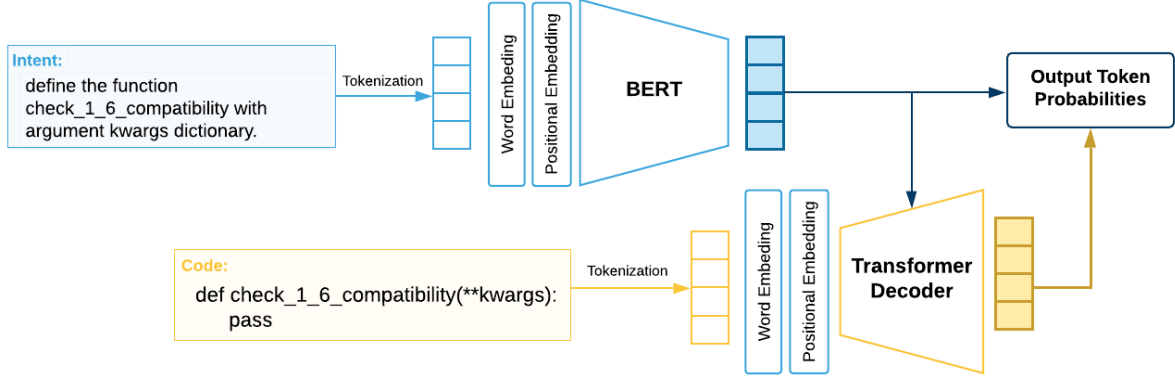
Figure 2: Model overview during training: we use a standard transformer-based encoder-decoder model where the positional and word embeddings are shared between encoder and decoder. The modules related to the encoder are represented in blue and the decoder ones are in yellow. Standard teacher forcing and transformer masking is applied during training.

The target y_i''s are reconstructed using the shared encoder-decoder model.

We conjecture that monolingual data auto-encoding mainly helps the decoder, so we propose to freeze the encoder parameters for monolingual data. Writing the encoder and decoder parameters separately with $\theta = [\theta_e, \theta_d]$, then θ_e is updated using the gradient of the supervised objective $\mathcal{L}_{\text{sup}}$, whereas the decoder gradient comes from $\mathcal{L}_{\text{full}}$. We verify this hypothesis in section 4.1.

In terms of model architecture, our TM is a standard transformer-based seq2seq model with copy attention (Gu et al., 2016) (illustrated in Fig. 2 of C). We fine-tune BERT as the encoder and use a 4-layer transformer decoder. There is little SP-specific inductive bias in the architecture. The only special structure is the copy attention, which is not a strong inductive bias designed for SP as copy attention is widely used in other tasks as well.

We refer to the method of using copied monolingual data and freezing the encoder over them as *target autoencoding* (TAE). Unless otherwise specified in the ablation studies, the encoder is always frozen.

4 Experiments

For our primary experiments we considered two python datasets namely Django and CoNaLa. The former is based on Django web framework and the latter is annotated code snippets from stackoverflow answers. Additionally, we experiment on the SQL version of GeoQuery and ATIS from Finegan-Dollak et al. (2018) (with query split), WikiSQL (Zhong et al., 2017), and Magic (Java) (Ling et al., 2016b).

Python Monolingual Corpora: CoNaLa comes with $600K$ mined questions from Stack-overflow. We ignored the noisy source intents/sentences and just use the python snippets. To be comparable with Xu et al. (2020), we also select a corresponding $100K$ subset version for comparison. See Appendix A for details on the SQL and Java monolingual corpora.

Experimental Setup: In all experiments, we use label smoothing with a parameter of 0.1 and Polyak averaging (Polyak and Juditsky, 1992) of parameters with a momentum of 0.999 except for GeoQuery which we use 0.995. We use Adam (Kingma and Ba, 2014) and early stopping based on the dataset specific evaluation metric on dev set. The learning rate for the encoder is 1×10^{-5} over all datasets. We used the learning rate of 7.5×10^{-5} on all datasets except GeoQuery and ATIS which we use $1 \times 10 - 4$. The architecture overview is shows in Fig. 2. At the inference time we use beam search with beam size of 10 and a length normalization based on (Wu et al., 2016). We run each experiment with 5 different random seeds and report the average and standard deviation. WordPiece tokenization is used for both natural language utterances and programming code.

4.1 Empirical Analysis

First, we considered a scenario where the monolingual corpus comes from the same distribution as the bitext. We simulate this setup by using 10% of Django training data as labeled data while using all the python examples from Django as the mono-

Source:	call the function lazy with 2 arguments : _string_concat and six.text_type , substitute the result for string_concat .	Source:	define the function timesince with d , now defaulting to none, reversed defaulting to false as arguments .
Gold & TAE:	```string_concat = lazy(_string_concat, six.text_type)```	Gold & TAE:	```def timesince(d, now=none, reversed=false): pass```
Baseline:	```string_concat = lazy (_concat_concat , six.text_type)```	Baseline:	```def timesince (d = none, reversed (d = false): pass```
Note:	copy mistake: wrong variable resulting from failed copy	Note:	unbalanced paranthesis and multiple semantic mistakes.

Table 1: Example mistakes by the baseline that are fixed by TAE. More examples in Appendix E.
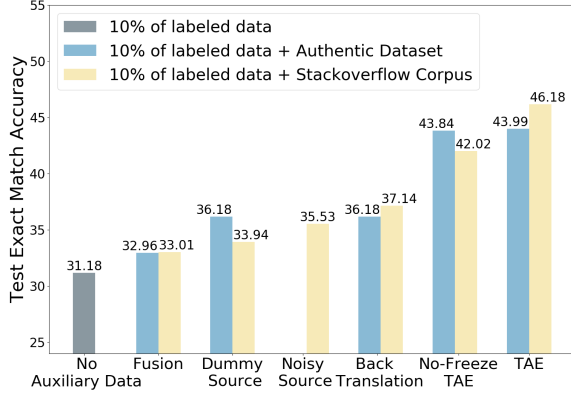

Figure 3: Analysis using only 10% Django train bitext.

lingual dataset of 10 times bigger. Results with "Authentic Dataset" in Fig. 3 shows the effectiveness of TAE vs other approaches.

Next, we used the monolingual dataset prepared for python (StackOverflow Corpus) which is from a different distribution. Fig. 3 shows even more considerable improvement, thanks to the larger monolingual set. We considered noisy intents provided in CoNaLa monolingual corpus and dummy source sentences where each monolingual sample is paired along with a random length array containing zeros. We also compared against other well-known approaches like fusion and back-translation, see experiments details in Appendix D. TAE outperforms all those approaches by a large margin.

Now one important question is, what part of the model benefits from monolingual data most? In Sec. 3.1, we conjectured that auto-encoding of monolingual data should mostly help the decoder, not the encoder. To verify this, we perform an ablation by comparing freezing encoder parameters versus not freezing over the monolingual set. Fig. 3 shows that without freezing the encoder, performance drops slightly for TAE on authentic Django while dropping significantly when copying on Stackoverflow data. This confirms that the performance gain is due to its effect on the decoder, while the copied monolingual data might even hurts the encoder.

4.2 Main Results on Full Data

Table 2-3 showcase our SOTA results on Django and CoNaLa. While our simple base seq2seq model does not outperform previous works, with TAE on the monolingual data, our performance improves and outperforms all the previous works.

The most direct comparison is with Xu et al. (2020) that also leverage the same extra data mined from StackOverflow (*EK* in Table 3). As mentioned in Sec. 2, they used the noisy parallel corpus for pre-training, whereas we only leverage the monolingual set. However, we obtain both larger relative improvements over our baseline (32.29 from 30.98) compared to Xu et al. (2020) (28.14 from 27.20), as well as better absolute results in the best case. In fact, with only the $100K$ StackOverflow monolingual data, our result is on par with the best one from Xu et al. (2020) that uses the additional python API bitext data. Finally, note that part of our superior performance is due to using BERT as an encoder.

Finally, TAE also yields improvements on other programming languages, as shown for GeoQuery (SQL), ATIS (SQL) and Magic (Java) in Table 4. We observe no improvement on WikiSQL. But it is not surprising given its large dataset size and the simplicity of its targets. As observed by previous works (Finegan-Dollak et al., 2018), more than half of queries follow simple pattern of "`SELECT col FROM table WHERE col = value`".

The main results in terms of improvement over previous best methods are statistically significant in Table 2-3. On Django, our result is better than Reranker (Yin and Neubig, 2019) (best previous method in Table 2) with a P-value < 0.05, under one-tailed two-sample t-test for mean equality. Since the previous state of the art on CoNaLa (EK + 100k + API in Table 3) did not provide the standard deviation, we cannot conduct a two-sample t-test against it. Instead, we performed a one-tailed two-sample t-test against the TranX+BERT baseline and observed that our improvement is statistically significant with P-value < 0.05. In Table 4,

Model	Django
YN17 (Yin and Neubig, 2017)	71.6
TRANX (Yin and Neubig, 2018)	73.7
Coarse2Fine (Dong and Lapata, 2018)	74.1
TRANX2 (Yin and Neubig, 2019)	77.3 ± 0.4
TRANX2 + BERT	79.7 ± 0.42
Reranker (Yin and Neubig, 2019)*	80.2 ± 0.4
Our baseline	77.05 ± 0.6
Our baseline + TAE	**81.03 ± 0.14**

Table 2: Exact match accuracy for Django test set. Yin and Neubig (2019)* trained a separate model on top of SP to rank beam search outputs.

Model	CoNaLa
Reranker (Yin and Neubig, 2019)*	30.11
TRANX (Yin and Neubig, 2019) + BERT	30.47 ± 0.7
EK (baseline) (Xu et al., 2020)	27.20
EK + 100k (Xu et al., 2020)	28.14
EK + 100k + API (Xu et al., 2020)*†	32.26
Our baseline	30.98 ± 0.1
Our baseline + TAE on 100k	32.29 ± 0.4
Our baseline + TAE on 600k	**32.57 ± 0.3**

Table 3: CoNaLa test BLEU. Methods with * trained a separate model on top of SP to rerank beam search outputs. Xu et al. (2020)† used an additional bitext corpus mined from python API documentation.

Dataset	Baseline (%)	Baseline + TAE (%)
GeoQuery	47.69 ± 0.05	51.87 ± 0.02
ATIS	38.04 ± 0.77	40.56 ± 0.57
Magic	41.61 ± 2.07	42.34 ± 0.52
WikiSQL	85.36 ± 0.06	85.30 ± 0.07

Table 4: Additional dataset results: test set exact match accuracy on all dataset.

improvements on GeoQuery and ATIS are statistically significant with P-value < 0.05, while it is not the case for Magic and WikiSQL.

4.3 Discussion

Thus far, we have verified that the decoder benefits from TAE and the encoder does not. For a better understanding of what TAE improves in the decoder, we propose two metrics namely *copy accuracy* and *generation accuracy*. Copy accuracy only considers tokens appearing in the source sentence. If the model produces all of the tokens that need to be copied from the source sentence, and in the right order, then the score is one otherwise zero for the example. Generation-accuracy ignores tokens appearing in the source intent and computes the exact match accuracy of the prediction. We show how to compute these metrics for the following example:

Question: define the function timesince with d, now defaulting to none, reversed defaulting to false as arguments.

Ground Truth:
"`def timesince(d, now=none, reversed=false): pass`"

We iterate over the ground truth script tokens one by one and remove those that can be copied from the source, leading to this code:

Generation Ground Truth:
"`def (=none=):pass`", and the removed tokens will be considered for copy ground truth.

Copy Ground Truth: "`timesince d , now , reversed false`".

We would then use the copy and generation ground truth strings to compute each metric. Note that the order of tokens are still important and exact equality is required.

As shown in Table 5 both metrics are improved. Table 1 illustrates one example from each type and with more samples in the Appendix E. Copy accuracy is important for producing the right variable names mentioned, and it is improved as expected. It is also encouraging to see quantitatively and qualitatively that grammar mistakes are reduced, meaning that the lack of prior knowledge of target language structure is compensated by learning from monolingual data.

Model	Copy	Generation
10% basline	34.18	55.73
10% baseline + TAE	58.89	66.31
Full baseline	80.11	81.27
Full baseline + TAE	84.59	82.65

Table 5: Copy and generation accuracies on Django test set

5 Conclusion

This work has shown the possibility to achieve a competitive or even SOTA performance on semantic parsing with little or no inductive bias design. Besides the usual large-scale pre-trained encoders, the key is to exploit relatively large monolingual corpora of the meaning representation. The modified copied monolingual data approach from machine translation literature works well in this extremely low-resource setting. Our results point to a promising alternative direction for future progress.

Acknowledgements

References

Miltiadis Allamanis and Charles Sutton. 2013. Mining Source Code Repositories at Massive Scale using Language Modeling. In *The 10th Working Conference on Mining Software Repositories*, pages 207–216. IEEE.

Franck Burlot and François Yvon. 2019. Using monolingual data in neural machine translation: a systematic study. *arXiv preprint arXiv:1903.11437*.

Franck Burlot and François Yvon. 2018. Using monolingual data in neural machine translation: a systematic study. *ArXiv*, abs/1903.11437.

John Cocke. 1969. *Programming languages and their compilers: Preliminary notes*. New York University.

Anna Currey, Antonio Valerio Miceli-Barone, and Kenneth Heafield. 2017. Copied monolingual data improves low-resource neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 148–156.

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*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*.

Jiatao Gu, Z. Lu, Hang Li, and V. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *ArXiv*, abs/1603.06393.

Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Huei-Chi Lin, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2015. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

J. Herzig and J. Berant. 2020. Span-based semantic parsing for compositional generalization. *arXiv preprint arXiv:2009.06040*.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24.

Tadao Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

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

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016a. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany. Association for Computational Linguistics.

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016b. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 574–584, Lincoln, Nebraska, USA. IEEE Computer Society.

Boris T Polyak and Anatoli B Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855.

Prajit Ramachandran, Peter J Liu, and Quoc V Le. 2016. Unsupervised pretraining for sequence to sequence learning. *arXiv preprint arXiv:1611.02683*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Anuroop Sriram, Heewoo Jun, S. Satheesh, and A. Coates. 2018. Cold fusion: Training seq2seq models together with language models. *ArXiv*, abs/1708.06426.

Felix Stahlberg, J. Cross, and Veselin Stoyanov. 2018. Simple fusion: Return of the language model. *ArXiv*, abs/1809.00125.

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

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Y. Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, M. Krikun, Yuan Cao, Q. Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, Taku Kudo, H. Kazawa, K. Stevens, G. Kurian, Nishant Patil, W. Wang, C. Young, J. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, G. S. Corrado, Macduff Hughes, and J. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

Frank F Xu, Zhengbao Jiang, Pengcheng Yin, Bogdan Vasilescu, and Graham Neubig. 2020. Incorporating external knowledge through pre-training for natural language to code generation. *arXiv preprint arXiv:2004.09015*.

Ziyu Yao, Daniel S Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1693–1703. International World Wide Web Conferences Steering Committee.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720*.

Pengcheng Yin and Graham Neubig. 2019. Reranking for neural semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4553–4559.

Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Jichuan Zeng, Xi Victoria Lin, S. Hoi, R. Socher, Caiming Xiong, Michael R. Lyu, and Irwin King. 2020. Photon: A robust cross-domain text-to-sql system. *ArXiv*, abs/2007.15280.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Datasets

We used 6 datasets in total. Django includes programs from Django web framework and CoNaLa contains diverse set of intents annotated on python snippets gathered from Stackoverflow. WikiSQL, GeoQuery, and ATIS include natural language questions and their corresponding SQL queries. WikiSQL includes single table queries while GeogQuery and ATIS requires queries on more than one table. Finally, Magic has Java class implementation of game cards with different methods used during the game. Table 6 summarises all the parallel datasets. For GeoQuery we used query split provided by (Finegan-Dollak et al., 2018).

Monolingual Corpus: CoNaLa comes with $600K$ mined questions from Stackoverflow. We ignored the noisy source intents/sentences and just use the python snippets. To be comparable with Xu et al. (2020), we also select a corresponding $100K$ subset version for comparison. For SQL, Yao et al. (2018) automatically parsed StackOverflow questions related to SQL and provided a set containing $120K$ SQL examples. We automatically parsed the SQL codes and removed samples with grammatical mistakes. We also filtered samples not starting with SELECT special token. Allamanis and Sutton (2013) downloaded full repositories of individual projects that were forked at least once; duplicate projects were removed. We randomly sampled 100K Java examples from more than 14K projects and use that as monolingual set. Table 7 summarises all the monolingual datasets.

Parallel Corpus	Language	Train	Dev	Test
Django (Oda et al., 2015) (link)	Python	16000	1000	1805
CoNaLa (Yin et al., 2018) (link)	Python	$2,179$	200	500
WikiSQL (Zhong et al., 2017) (link)	SQL	$56,355$	8421	15878
ATIS (Finegan-Dollak et al., 2018) (link)	SQL	4812	121	347
GeoQuery (Finegan-Dollak et al., 2018) (link)	SQL	536	159	182
Magic (Ling et al., 2016a) (link)	Java	$8,457$	446	483

Table 6: Parallel dataset sizes. We filtered out Magic data with java code longer than 350 tokens in order to fit in GPU memory.

Monolingual Corpus	Source	Size
Python (Yin et al., 2018) (link)	Stackoverflow	100K
SQL (Yao et al., 2018) (link)	Stackoverflow	52K
Java (Allamanis and Sutton, 2013) (link)	Github	100k

Table 7: Monolingual dataset sizes.

B Dev Set Results

Dataset	Baseline (%)	Baseline + TAE (%)
CoNaLa	32.43 ± 0.21	34.81 ± 0.36
ATIS	5.79 ± 0.29	7.23 ± 0.45
GeoQuery	53.33 ± 1.47	52.58 ± 0.70
Django	75.52 ± 0.21	78.56 ± 0.39
Magic	42.26 ± 1.42	44.17 ± 0.99
WikiSQL	85.92 ± 0.09	85.83 ± 0.07

Table 8: Dev set exact match accuracy on all datasets except CoNaLa which uses BLEU. We followed (Yin and Neubig, 2018) implementation of BLEU score which can be found here.

C Architecture and Experiment Details

We selected the decoder learning rate based on linear search over $[1 \times 10^{-3} - 2.5 \times 10^{-5}]$. Number of decoder layers has been decided based on search over $\{2, 3, 4, 5, 6\}$ layers and 4 layer decoder shows superior performance (we used a single run for hyperparameter selection). Each model has 150M parameters optimized using a single GTX 1080 Ti GPU. With batch size of 16 each step takes 1.7s on GeoQuery dataset (other datasets have very similar runtime). On Django and CoNaLa, we followed (Yin and Neubig, 2018; Xu et al., 2020) on replacing quoted values with a "str#" where # is a unique id. On Magic dataset, we replaced all newline "\n" tokens with "#"; following (Ling et al., 2016a), we splitted Camel-Case words (e.g., class TirionFordring → class Tirion Fordring) and all punctuation characters. We filtered out Magic data with java code longer than 350 tokens in order to fit in GPU memory.

D Back-Translation and Fusion details

For fusion we follow equation 1 where TM stands for translation model and LM stands for language model. τ limits the confidence of the language model and λ controls the balance between TM and LM. figure 4 shows the performance of a base TM trained on 10% of Django training data with test exact match accuracy of 31.80 over different values of λ and τ. The LM is trained over full Django training set.

$$\log p(y_i^t) = \log p_{TM}(y_i^t) + \lambda \log p_{LM}(y_i^t) = \log p_{TM}(y_i^t) + \lambda \log \frac{e^{l_i^t/\tau}}{\sum_i e^{l_i^t/\tau}} \quad (1)$$
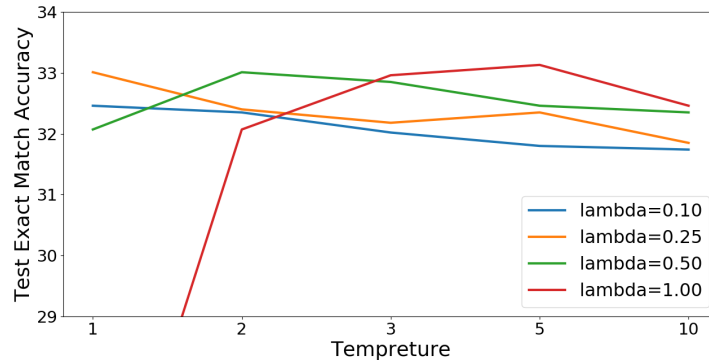

Figure 4: Test exact match accuracy of TM leverage fusion with different parameters

For back-translation we first trained the model using the same architecture explained above in the backward direction. We used BLEU score as a evaluation metric and use early stopping based on that. Using greedy search we generate the corresponding source intent for each code snippet. In the end, the synthetic data is merged with the bitext and trained a forward model.

E   Additional Qualitative Examples

Source:	`call the function lazy with 2 arguments : _string_concat and six.text_type [ six . text_type ] , substitute the result for string_concat .`
Gold:	`string_concat = lazy(_string_concat, six.text_type )`
Baseline:	`string_concat = lazy (``_concat``_concat , six. text_type )`
TAE:	`string_concat = lazy ( _string_concat , six. text_type )`
Note:	wrong var

Source:	`get translation_function attribute of the object t , call the result with an argument eol_message , substitute the result for result .`
Gold:	`result = getattr(t, translation_function)( eol_message)`
Baseline:	`result = getattr ( t , translation_``message`` ) ( eol_message )`
TAE:	`result = getattr ( t , translation_function ) ( eol_message )`
Note:	wrong var

Source:	`convert whitespace character to unicode and substitute the result for space .`
Gold:	`space = unicode(' ')`
Baseline:	`space = unicode ( ``character`` )`
TAE:	`space = unicode ( ' ' )`
Note:	wrongly copied variable name

Source:	`assign integer 2 to parts if third element of version equals to zero , otherwise assign it integer 3 .`
Gold:	`parts = 2 if version[2] == 0 else 3`
Baseline:	`parts ``[ 2 ] = 2`
TAE:	`parts = 2 if version [ 2 ] == 0 else 3`
Note:	baseline failed to copy a few source tokens, and instead formed a grammatically correct but semantically incorrect output

Copy mistake examples

Source:	`define the function timesince with d , now defaulting to none , reversed defaulting to false as arguments .`
Gold:	`def timesince(d, now=none, reversed=false): pass`
Baseline:	`def timesince ``( d = none, reversed ( d = false ) :`` pass`
TAE:	`def timesince ( d, now = none, reversed = false ) : pass`
Note:	unbalanced paranthesis and multiple semantic mistakes.

Source:	`define the function exec with 3 arguments : _code_ , _globs_ set to none and _locs_ set to none .`
Gold:	`def exec_(_code_, _globs_=none, _locs_=none): pass`
Baseline:	`def exec ( _code_ , ``_globs``= none , ``_locs_ set ( ) )`` : pass`
TAE:	`def exec ( _code_ , _globs_ = none , _locs_ = none ) : pass`
Note:	wrong variable name and grammar mistake

Source:	`return an instance of escapebytes , created with an argument , reuslt of the call to the function bytes with an argument s .`
Gold:	`return escapebytes(bytes(s))`
Baseline:	`return escapebytes ( bytes ( s ) ``. re ( s )`
TAE:	`return escapebytes ( bytes ( s ) )`
Note:	extra semantically incorrect predictions and unbalanced paratheses

Source:	`call the function blankout with 2 arguments : p and str0 , write the result to out .`
Gold:	`out.write(blankout(p, 'str0'))`
Baseline:	`out .write ( blankout ( p , 'str0' ``)`
TAE:	`out .write ( blankout ( p , 'str0' ) )`
Note:	unbalanced paratheses

Grammar or semantic mistake examples

Table 9: Mistake examples