EXPLICIT KNOWLEDGE TRANSFER FOR WEAKLY-SUPERVISED CODE GENERATION

Zhangir Azerbayev, Ansong Ni, Hailey Schoelkopf, Dragomir Radev
Yale University
zhangir.azerbayev@yale.edu

ABSTRACT

Large language models (LLMs) can acquire strong code-generation capabilities through few-shot learning. In contrast, supervised fine-tuning is still needed for smaller models to achieve good performance and such fine-tuning demands a large number of task-specific NL-code pairs, which are expensive to obtain. In this paper, we attempt to transfer the code generation ability of an LLM to a smaller model with the aid of weakly-supervised data. More specifically, we propose *explicit knowledge transfer* (EKT), which uses the few-shot capabilities of a teacher LLM to create NL-code pairs that we then filter for correctness and fine-tune the student on. We evaluate EKT on the task of generating code solutions to math word problems from the GSM8k dataset. We find that EKT not only yields better performance than training with expert iteration, but also outperforms knowledge distillation, another form of knowledge transfer. A GPT-Neo 1.3B model trained using EKT with a GPT-J teacher achieves a 12.4% PASS@100 on GSM8k, while the same student and teacher trained with knowledge distillation yield only a 3.7% PASS@100. We also show that it is possible for a student model to outperform the teacher using EKT.

1 INTRODUCTION

Code generation is the task of solving problems described in natural language (NL) by generating and subsequently executing code solutions written in a general-purpose programming language. Pretrained language models have demonstrated impressive code generation capabilities within two different paradigms. In *few-shot learning*, a model is conditioned to generated code by inserting a small number of NL-code pairs in its context. This method enables flexible generalization to new tasks, but only performs well with large language models (LLMs) that are computationally expensive at inference (Brown et al., 2020; Wei et al., 2022). The other paradigm, *supervised fine-tuning*, can achieve strong performance with smaller models. However, fine-tuning requires large amount of labelled training data, which is costly to manually collect and annotate (Chen et al., 2021; Xu et al., 2022).

We look to training with weak supervision as a way to achieve strong performance with small models whilst alleviating the data collection needs of supervised fine-tuning. Training with weak supervision means that only the natural language input and the expected execution result are provided for learning, leaving the gold programs latent (Pasupat & Liang, 2015). Though such weakly-supervised training data is cheaper to obtain, the learning signal from it can be weak and noisy (Ni et al., 2020).

In this paper, we propose *Explicit Knowledge Transfer* (EKT), a method that trains a language model on weakly-supervised data with the aid of a black-box teacher LLM. In particular, EKT uses the few-shot capabilities of an LLM to create NL-code pairs that we then filter for correctness and fine-tune on. EKT retains the compute and memory advantages of small models at inference while avoiding the need to create large fully-labelled datasets. We compare EKT to knowledge-distillation (Hinton et al., 2015; Sanh et al., 2019), an alternative method for leveraging a teacher model that trains the output distribution of the student against the output distribution of the teacher. Additionally, we evaluate EKT against expert iteration (Silver et al., 2017; Polu et al., 2022), which is a method that directly learns from weak supervision.

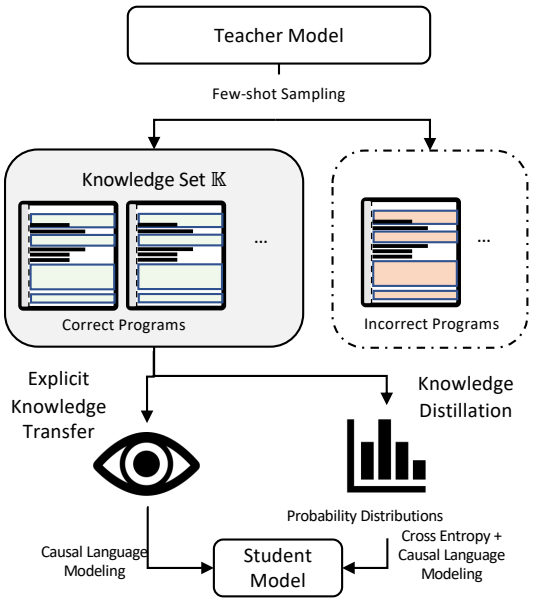

Figure 1: Schematic depiction of knowledge distillation (right) and explicit knowledge transfer (left) for weakly-supervised code generation.

Results on the Grade-school-math (GSM8k) (Cobbe et al., 2021) dataset show that EKT not only improves the performance of a student model learned with weak supervision, but also yields better performance than knowledge distillation from the same teacher LLM. By transferring from a GPT-J 6B model (Wang & Komatsuzaki, 2021), EKT is able to boost the performance of a GPT-Neo 1.3B (Black et al., 2021) student from 2% pass@100 to 12.4%, while knowledge distillation only marginally improves it to 3.7%. We also perform an ablation study for EKT by varying the model sizes for both the student and the teacher. Our best performance is achieved with EKT by transferring from the Codex model (Chen et al., 2021) to GPT-Neo 1.3B, which boosts the performance to 32.9% pass@100.

2 Methodology

We describe the weakly-supervised code generation task in § 2.1 and existing methods for tackling the problem in § 2.2. We introduce knowledge transfer methods, including EKT in § 2.3.

2.1 Problem Formulation

Let x be an NL specification and y^* be the gold program that satisfies this specification by achieving the desired execution result z. We also assume access to a known, task-specific boolean function $f(y, z)$ that verifies the the correctness of a candidate program y against z (*e.g.,* test cases). Weakly-supervised code generation is the task of learning a parameterized model $P(y|x; \theta)$ from (x, z) pairs, leaving y^* latent. In practice, we seed our weakly-supervised learner at the beginning with a small set of NL-code examples $S = \{(x_1, y_1), \dots\}$ to ensure nontrivial gradient updates.

2.2 Existing Methods

Expert Iteration. Since $P(y|x; \theta)$ cannot be directly optimized, much work in weakly-supervised learning has relied on *expert iteration* (Silver et al., 2017; Liang et al., 2017; Polu et al., 2022). In expert iteration, a model is seeded with an initial set of NL-code pairs D_0. Training proceeds by alternating two steps: 1) a sampling step, where we sample candidate solutions to weakly-supervised training examples from the model, then filter for correctness and 2) a training step, where the model is trained with MLE on all known correct NL-code pairs.

Few-shot Learning. Another alternative when few NL-code pairs are available is to ignore weakly-supervised data altogether, and do few-shot learning by formatting the examples in S as a prompt. This method typically does not achieve strong performance with small models (Austin et al., 2021; Wei et al., 2022).

2.3 KNOWLEDGE TRANSFER

In expert iteration, the same model both generates novel correct programs and learns from these correct programs. We observe that generating novel samples is the step that bottlenecks the performance of such methods, which motivates offloading generation to a teacher LLM with few-shot learning abilities. In this section, we describe how to transfer code generation ability from a teacher model $P(y|x, \theta_t)$ to a student model $P(y|x, \theta_s)$.

Knowledge Set Acquisition. The following describes a procedure for creating the *knowledge set* $\mathbb{K}$ of a teacher LLM from a weakly-supervised dataset $\mathcal{D} = \{(x_i, z_i)\}_{i=1}^{|\mathcal{D}|}$ and some few-shot examples $S = \{(x_i, y_i\}_{i=1}^{|S|}$, where $|S|$ is small. For each $(x, z) \in \mathcal{D}$, perform few-shot learning with S and sample code solutions $C = \{\hat{y}_1, ...\}$ from the teacher model, *i.e.,* $\hat{y}_i \sim P(y|x, S; \theta_t)$. Then choose one $\hat{y}^* \in C$ such that $f(\hat{y}^*, z) = 1$, meaning $\hat{y}^*$ is a solution for x, if such a $\hat{y}^*$ exists. [1] Collect each $(x, \hat{y}^*, z)$ into a *knowledge set* $\mathbb{K}$.

Explicit Knowledge Transfer. Given a knowledge set $\mathbb{K}$ from the teacher model, EKT with a weakly-supervised dataset $\{(x, z)\}$ proceeds as follows. For each NL-code pair (x, y, z) in $\mathbb{K}$, we concatenate x and $\hat{y}$ into a single sequence of tokens $t = (t_1, \ldots, t_n)$ and train the student model $P(y|x; \theta_s)$ using a causal language modeling (CLM) objective, following recent work on code generation (Chen et al., 2021; Austin et al., 2021; Ni et al., 2022):

$$\begin{aligned}\mathcal{L}_{\text{MLE}}(t, \theta_s) &= \sum_{k=1}^{n} H(P(t_k|t_{<k}; \theta_s), e_{t_k}) \\ &= -\sum_{k=1}^{n} \log P(t_k|t_{<k}; \theta_s)\end{aligned}$$

Here H denotes cross entropy and e_{t_k} is the one-hot encoding of token y_k.

Knowledge Distillation. Knowledge distillation refers to a broad class of knowledge transfer methods whose loss incorporates the cross entropy between the student's output distribution and a teacher' output distribution. Knowledge distillation has emerged as one of the standard methods for knowledge transfer, as the non-modal probability masses of $P(y|x; \theta_t)$ are thought to encode rich information about the generalization power of the teacher (Hinton et al., 2015; Papernot et al., 2015; Sanh et al., 2019)

We compare EKT to a knowledge distillation baseline trained on the same knowledge set $\mathbb{K}$. In particular, define the *distillation loss*:

$$\mathcal{L}_{\text{CE}}(t, \theta_t, \theta_s) = \sum_{k=1}^{n} H(P(t_k|t_{<k}; \theta_s), P(t_k|t_{<k}; \theta_t))$$

Following Sanh et al. (2019), we train our knowledge distillation student θ_s on a loss of the form $\mathcal{L} = \alpha\mathcal{L}_{\text{CE}} + (1 - \alpha)\mathcal{L}_{\text{MLE}}$ where $0 < \alpha < 1$.

Note that knowledge distillation is limited to the case where the teacher and the student share a vocabulary, while EKT is not.

[1] If a teacher generates multiple programs $\hat{y}$ passing the test cases z for a training example x, we do not use any heuristics to decide which program is selected as $\hat{y}*$, and simply choose one uniformly at random.

Model	PASS@1	PASS@100
Few-shot learning:		
Minerva 540B [a*]	58.8	-
PaLM 540B[b*]	56.5	-
LaMDA 137B[c*]	7.6	-
Codex-Davinci	37.4	92.3
Codex-Cushman	5.0	58.0
GPT-Neo 1.3B	1.4	2.0
Fine-tuned w/ full supervision:		
GPT-3 6B + verifier[d*]	39.0	-
GPT-3 175B + verifier[d*]	55.0	-
GPT-Neo 2.7B + SS[e]	19.5	41.4
Weakly-supervised methods (all w/ GPT-Neo 1.3B):		
Expert Iteration	0.0	0.0
KD from GPT-J 6B	0.1	3.7
EKT from GPT-J 6B	1.8	12.4
EKT from Codex-davinci	**18.2**	**32.9**

Table 1: Comparison of EKT to baseline methods on test set of GSM8k. Note that we are unable to evaluate KD from Codex-davinci to GPT-Neo, since the models have different vocabularies. [a]: Lewkowycz et al. (2022) [b]: Chowdhery et al. (2022); [c]: Thoppilan et al. (2022); [d]: Cobbe et al. (2021); [e]: SS denotes self-sampling from Ni et al. (2022); *: models that generate NL solutions instead of code; -: no results are available.

3 EXPERIMENTS

3.1 SETUP

Dataset. We conduct experiments on the GSM8k dataset (Cobbe et al., 2021), which consists of 7.4K training examples of grade-school-level math questions and their numerical answers[2]. We approach this problem by generating Python code solutions and attempt to learn the model with weak supervision.

Models and Training. We choose GPT-Neo models (Black et al., 2021) of sizes 125M and 1.3B as student models, with GPT-Neo 2.7B, GPT-J 6B (Wang & Komatsuzaki, 2021) and Codex[3] (Chen et al., 2021) as ablations for different teacher models. In our experiments, we evaluate four learning methods: 1) few-shot learning; 2) expert iteration (EI); 3) knowledge distillation (KD); and 4) explicit knowledge transfer (EKT). For EI, we initialize D_0 by using a few-shot prompt to sample from our not yet fine-tuned model $P(y|x, S; \theta)$, followed by correctness filtering.

Evaluation Metric. We evaluate code generation performance using the PASS@k metric (Chen et al., 2021). Given k sampled programs for a given NL specification, PASS@$k = 1$ if any of the programs $\hat{y}$ is correct (*i.e.,* $f(\hat{y}, z) =$ `True`), then the average PASS@k is reported.

More details for the experimental setups (*e.g.,* hyperparameters, few-shot prompts) can be found in Appendix A.

3.2 KNOWLEDGE TRANSFER RESULTS

Effectiveness of EKT. In Tab. 1, we compare to performance of EKT to various baselines. EKT outperforms all other weakly-supervised training methods. We offer a hypothesis as to why EKT outperforms knowledge distillation despite learning from the same knowledge set. Because we sample from the teacher at a high temperature to generate the knowledge set, it may be that the teacher generates a correct program by sampling from a region of low probability mass. Since EKT trains with MLE, the student will "disagree" with the teacher and assign this generated program a

[2]The original dataset also contains NL solutions but we ignore them in our setting since they are not executable.

[3]More specifically, we the use `code-davinci-002` engine.

Teacher model Student model	Coverage	PASS@1	PASS@100
GPT-Neo 2.7B	19.5	0.4	13.0
GPT-Neo-125M	-	0.3	10.8
GPT-Neo-1.3B	-	1.1	8.7
GPT-J 6B	39.7	0.8	30.6
GPT-Neo-125M	-	1.1	13.6
GPT-Neo-1.3B	-	1.8	12.4
Codex-Davinci	93.0	37.4	92.3
GPT-Neo-125M	-	5.6	18.2
GPT-Neo-1.3B	-	18.7	32.9

Table 2: Ablation study demonstrating the effect of modulating the teacher and the student when training with EKT. *Coverage* is the percentage of training examples where a teacher generated a correct program.

high probability. In contrast, knowledge distillation will mimic the output distribution of the teacher, and thus assign a low probability to all programs the teacher assigned a low probability to.

Despite the strong performance of expert iteration in other settings, it does not achieve nontrivial performance in our experiments. The poor performance of expert iteration is explained by the fact that the seed dataset D_0 is too small for the model to achieve non-trivial bootstrapping ability. For example, in expert iteration with GPT-Neo 1.3B, we see that $|D_0| = |D_1| = 43$, suggesting the model is unable to generalize when fine-tuning on only 43 examples.

3.3 Ablation on Student and Teacher Models

In Tab. 2 we present an ablation study that demonstrates the effect of modulating the size of the teacher and of the student. We observe that when using EKT, increasing the strength of the teacher is more effective than increasing the capacity of the student. For example, using a GPT-Neo 125M model with a GPT-J 6B teacher as a baseline, switching to a GPT-Neo 1.3B student only improves PASS@1 by 0.7% while upgrading to a Codex teacher increases PASS@1 by 4.4% and PASS@100 from 12.4% to 18.2%. We believe this is because more capable teacher provides better coverage (*i.e.,* 39.7% and 93.0% in this case), which enables the student to learn from more training examples. The hypothesis is supported by the fact the 125 million parameter student has a PASS@1 close to the 1.3 billion parameter student and a slightly higher PASS@100 when transferring from a GPT-J teacher using EKT. In this case, increasing the capacity of the student in a low-data regime may lead to a degraded PASS@100 because a stronger model overfits more easily and thus does not preserve sample diversity.

In the case of GPT-Neo 2.7B and GPT-J, both student models outperform their EKT teacher at PASS@1. This is not unexpected for two reasons. First, the student model learns from correct solutions generated by the teacher in the setting of a high sampling budget and a high sampling temperature. This means that the student can learn from programs that the teacher is unlikely to generate in the setting of low-temperature or greedy decoding. Second, the student is conditioned on more task specific data, namely the teacher's knowledge set.

4 Related Work

Language models trained on code. Code LMs have achieved strong code generation performance with few-shot learning in the case of LLMs and supervised fine-tuning in the case of small models (Chen et al., 2021; Austin et al., 2021; Li et al., 2022; Nijkamp et al., 2022; Fried et al., 2022).

Weakly-supervised semantic parsing. Semantic parsing is the task of mapping an NL utterance to an executable formal representation, possibly not in a general purpose language, and the weakly-supervised case is a well-studied problem in NLP (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2012; Zhong et al., 2017; Liang et al., 2017; Dong & Lapata, 2018).

Knowledge Transfer *Knowledge distillation* is a popular method of knowledge transfer for neural networks (Hinton et al., 2015; Papernot et al., 2015; Sanh et al., 2019).

5 Conclusion

We proposed EKT, a method of weakly-supervised code generation that leverages a teacher LLM. We showed that EKT outperforms weakly-supervised baselines and other knowledge transfer methods, with a GPT-Neo 1.3B student trained using EKT with a Codex-davinci teacher achieving a PASS@1 of 18.2% on GSM8k. Additionally, we studied the effects of modulating the size of the student and the teacher model.

Acknowledgements

We would like to thank OpenAI for providing access to the Codex API public beta.

References

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cogn. Sci.*, 9:147–169, 1985.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large scale autoregressive language modeling with mesh-tensorflow, 2021. URL http://github.com/eleutherai/gpt-neo.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean,

Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 731–742, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1068. URL https://aclanthology.org/P18-1068.

Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation, 2017. URL https://arxiv.org/abs/1707.02633.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis, 2022. URL https://arxiv.org/abs/2204.05999.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode, 2022. URL https://arxiv.org/abs/2203.07814.

Chen Liang, Jonathan Berant, Quoc V. Le, Ken Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23–33, Vancouver, Canada, 2017. URL http://aclanthology.coli.uni-saarland.de/pdf/P/P17/P17-1003.pdf.

Ansong Ni, Pengcheng Yin, and Graham Neubig. Merging weak and active supervision for semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8536–8543, 2020.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Oleksandr Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. Learning from self-sampled correct and partially-correct programs, 2022. URL https://arxiv.org/abs/2205.14318.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. A conversational paradigm for program synthesis, 2022. URL https://arxiv.org/abs/2203.13474.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2015. URL https://arxiv.org/abs/1511.04508.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. 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)*, pp. 1470–1480, 2015.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning, 2022. URL https://arxiv.org/abs/2202.01344.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019. URL https://arxiv.org/abs/1910.01108.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL https://arxiv.org/abs/1712.01815.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications, 2022. URL https://arxiv.org/abs/2201.08239.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent J Hellendoorn. A systematic evaluation of large language models of code. *arXiv preprint arXiv:2202.13169*, 2022.

John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pp. 1050–1055. AAAI Press, 1996. ISBN 026251091X.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars, 2012. URL https://arxiv.org/abs/1207.1420.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. URL https://arxiv.org/abs/1709.00103.

A HYPERPARAMETERS

A.1 TRAINING

We use the hyperparameters in Tab. 4 to train our student models, both in the EKT and knowledge distillation case.

For expert iteration, each M_n is trained using the same hyperparameters as in table $Tab.\ 4$, except we train for 4 epochs and use a fixed learning rate of $5 \cdot 10^{-5}$. We stop expert iteration at the iteration N where $|\mathbb{K}_{N-1}| = |\mathbb{K}_N|$. After stopping iteration, we further train M_N on $\mathbb{K}_N$ using the hyperparameters in table Tab. 4 to yield our final trained model.

For knowledge distillation, we use a weighting factor of $\alpha = 1/2$.

For our experiments, we reserve 500 randomly-sampled training examples as a validation set, and so do not train our models on those examples.

```
# The total average age of three friends is 40.  Jared is ten years older than Hakimi, and Molly's age is 30.
How old is Hakimi?
n0 = 40
n1 = 10
n2 = 30
t0 = 3 * n0
t1 = t0 - n2
answer = (t1 - n1) / 2

# A carpenter worked alone for 1 day on a job that would take him 7 more days to finish.  He and another car-
penter completed the job in 4 more days.  How many days would it have taken the second carpenter to do the
complete job working alone?
n0 = 1.0
n1 = 7.0
n2 = 4.0
t0 = n0 + n1
t1 = n2 * t0
answer = t1 / 2.0

# In two alloys, copper and tin are related in the ratios of 4 :  1 and 1 :  3.  10 kg of 1st alloy, 16 kg of
the 2nd alloy and some pure copper are melted together.  An alloy is obtained in which the ratio of copper and
tin was 3 :  2 .  Find the weight of the new alloy.
n0 = 4.0
n1 = 1.0
n2 = 1.0
n3 = 3.0
n4 = 10.0
n5 = 16.0
n6 = 2.0
n7 = 3.0
n8 = 2.0
t0 = n4 + n5
t1 = n0 + n1
t2 = n3 / n0
t3 = n4 / t1
t4 = n5 * t2
t5 = t3 + t4
t6 = n3 * t5
t7 = t6 / n6
t8 = t7 - t4
answer = t0 + t8
```

Table 3: Our few-shot prompt S.

Parameter	Setting
Training Epochs	140
Learning Rate (LR)	$1 \cdot 10^{-4}$
Optimizer	AdamW
Adam Betas	$(0.9, 0.999)$
Adam Eps	$1 \cdot 10^{-8}$
Weight Decay	0.1
LR Scheduler	Linear w/ warm-up
LR Warm-up Steps	100
Effective Batch Size	32
Precision	FP32
Gradient Clipping	1.0

Table 4: Student training hyperparameters.

A.2 Sampling and Evaluation

We use the few shot prompt in table Tab. 3. To generate the knowledge set $\mathbb{K}$ from our teacher LLMs, we generate 100 samples per training example using temperature sampling at a temperature of 0.6 (Ackley et al., 1985; Ficler & Goldberg, 2017). To calculate PASS@1 on the test set, we generate samples with greedy decoding and for PASS@100 on the test set, we generate samples using temperature sampling with a temperature of 0.6.

For expert iteration, we create the initial knowledge-set D_0 by sampling from our not yet fine-tuned model $P(y|x, S; \theta)$ with 100 samples per question and a temperature of 0.6 on the GSM8k training set and filtering for correctness.

Teacher model	**Student model**	**Student training method**	Teacher			Student	
			training set coverage	**PASS@1**	**PASS@100**	**PASS@1**	**PASS@100**
None	GPT-Neo 125M	Few-shot	-	-	-	0.2%	0.4%
None: GPT-Neo 2.7B		EI	-	-	-		
GPT-Neo 2.7B		KD	19.5%	0.4%	13%	0%	2%
GPT-Neo 2.7B		EKT	19.5%	0.4%	13%	0.3%	10.8%
GPT-J		KD	39.7%	0.8%	30.6%	0%	3.6%
GPT-J		EKT	39.7%	0.8%	30.6%	1.1%	13.6%
Davinci-code-002		EKT	93.0%	37.4%	92.3%	5.6%	18.2%
None	GPT-Neo 1.3B	Few-shot	-	-	-	1.4%	
None		EI	-	-	-		
GPT-Neo 2.7B		KD	19.5%	0.4%	13%	0%	2.8%
GPT-Neo 2.7B		EKT	19.5%	0.4%	13%	1.1%	8.7%
GPT-J		KD	39.7%	0.8%	30.6%	0.1%	3.7%
GPT-J		EKT	39.7%	0.8%	30.6%	1.8%	12.4%
Davinci-code-002		EKT	93.0%	37.4%	92.3%	18.7%	32.9%

Table 5: Complete results of our experiments on the GSM8k dataset. EI: expert iteration; KD: knowledge distillation. *Training set coverage* denotes the percentage of training examples where the teacher generated at least one correct program during sampling.

B COMPLETE RESULTS

For complete results of all our experiments, see Tab. 5.