Prompt2DeModel: Declarative Neuro-Symbolic Modeling with Natural Language

Hossein Rajaby Faghihi[1]**, Aliakbar Nafar**[1]**, Andrzej Uszok**[2]**, Hamid Karimian**[1]**, and Parisa Kordjamshidi**[1]
[1]**Michigan State University** [2]**Florida Institute for Human and Machine Cognition**
`{rajabyfa, nafarali, karimian, kordjams}@msu.edu, auszok@ihmc.org`

Abstract. This demo presents a conversational pipeline for crafting complex neuro-symbolic models through natural language prompts. It leverages large language models to generate declarative programs in the DomiKnowS framework. The programs in this framework express concepts and their relationships as a graph in addition to logical constraints between them. The graph will be connected to underlying neural models. Our proposed pipeline utilizes techniques like dynamic in-context demonstration retrieval, model refinement based on feedback from a symbolic parser, visualization, and user interaction to generate the tasks' structure and formal knowledge representation. This approach empowers domain experts, even those not well-versed in ML/AI, to design customized neural models that can incorporate their respective domain knowledge.

1 Introduction

With the rise of large language models (LLM), the community is getting closer to the dream of natural language programming interfaces [22, 27]. Nonetheless, developing systems capable of seamlessly and accurately interpreting and executing complex programming tasks using natural language remains a challenging goal. Conversely, declarative programming has endeavored to maintain programs at the problem domain level [18]. Within this programming paradigm, emphasis on the logic of what needs to be accomplished, as opposed to the intricacies of implementation, reduces the gap between natural and declarative languages. Inspired by this idea, our research envisions a natural language interface for a *declarative learning-based programming* framework [26, 15]. Within the interface, developers articulate their learning tasks using natural language, where the underlying system translates them into code in the declarative language of DomiKnowS[1] [7]. DomiKnowS further establishes connections between symbolic knowledge and neural learning components, facilitating the integration of domain knowledge into deep learning models.

The need for developing customized learning-based models is driven by two main arguments: 1) Despite their impressive performance across various tasks [37], Large Language Models (LLMS) still trail behind smaller, domain-specific models [32, 31, 14], and 2) incorporating domain knowledge into learning models can improve their performance or ensure output consistency with the domain knowledge [23, 25, 34].

To develop our method, we leverage the progress on specialized frameworks for knowledge integration [7, 1, 19, 12], particularly those with declarative interfaces [16] and the advancements in tailoring neural model architectures using LLMs [31]. Our method offers two advantages over prior research on tailoring deep learning models from natural language prompts. Firstly, we progress towards developing complex architectures, in contrast to previous research focusing on simply fine-tuning text-to-text models for specific tasks [31]. Secondly, our approach aids in fine-tuning complex architectures, which distinguishes it from prior work primarily centered on the composition of pre-trained models during inference [29].

Adapting LLMs to generate customized neural architectures based on the DomiKnowS framework presents several challenges. These include limited resources for fine-tuning, the requirement to adjust LLMs to an unfamiliar output format, the complexity of translating domain knowledge into FOL statements [10], let alone the specific DomiKnowS' Python-based First-Order Logic (FOL) language, and the potential lack of user familiarity with the DomiKnowS language, hindering meaningful feedback in a human-in-the-loop process. To tackle these issues, we propose an interactive pipeline where we utilize underlying LLMs with techniques such as prompt templates for user interactions, dynamic few-shot in-context learning, intermediary mapping of natural language (NL) to FOL statements, and iterative refinement based on feedback from symbolic semantic/syntactic verification functions. The demo[2] and the video[3] can be found online.

2 Related Research

Our proposal aims to facilitate the development of neural models leveraging background knowledge expressed in logical constraints. This approach, explored both during inference [8, 28, 9] and training [11, 20, 33] of neural networks, has been encapsulated in libraries like DeepProbLog [19], PyLon [1], and Scallop [12]. DomiKnowS [7] provides a declarative interface for defining knowledge and computational units, allowing seamless integration of knowledge using various techniques. While DomiKnowS represents a significant

[1] https://github.com/HLR/DomiKnowS

[2] To access the demo, use the following user: "Reviewer" and Password: "DemoReviewer2024" at https://hlr-demo.egr.msu.edu/.

[3] https://youtu.be/9q9PvH3dJKE

advancement, our proposal seeks to enhance the declarative interface by enabling the direct use of task descriptions in natural language.

Moreover, existing endeavors in code generation [5, 30] deploy large models in few-shot or zero-shot settings [36, 3] or fine-tune them for specific tasks [22, 27]. These approaches rely on encountering similar data in pre-training or demand substantial data resources for fine-tuning, which are inaccessible to us. Our work emphasizes using natural language interfaces to generate code that supports the development of neural models instead of raw code in popular programming languages.

Recent research has investigated generating neural architecture from natural language prompts by composing models during inference [29], generating training data [35], or training straightforward architectures [31]. Differently, we facilitate mapping natural language prompts to formal symbolic representations beneficial for crafting complex declarative learning-based models.

Our proposal aligns with recent research on the trajectory of deriving formal representations from language models, where recent research has delved into such translation to formalism suitable for underlying engines for logical inference and constraint optimization domains [23] and guided generation for consistent reasoning [24]. However, we propose an interactive pipeline for extracting representations, facilitating the development of neural architectures intricately connected to symbolic concepts. Our task includes an additional complexity, as the target formalism is not seen during LLMs' pretraining.

Lastly, our work employs techniques for sampling LLMs' output, emphasizing that correct response can be obtained with a sufficient number of samples [21]. Techniques like majority voting, post-pruning, and filtering by test cases have enhanced accuracy in large-scale sampling [4, 2, 6]. Due to the dynamic nature of tasks and neural models, defining test cases is impossible for our use case. Instead, we employ an iterative feedback system that provides execution errors to the language model to reflect on. In contrast to self-refinement research [13], our work introduces an external symbolic parser for the semantic evaluation of expected output structures, offering insights for iterative error correction.

3 DomiKnowS Interface

Our tool aims to translate natural language descriptions of structured prediction tasks into corresponding representations within the DomiKnowS framework. DomiKnowS programs include three components: knowledge declaration (task structure and constraints), model declaration (computational units), and program execution (knowledge integration, learning, and inference). This paper focuses on the knowledge declaration stage, as it often requires input from domain experts who may lack familiarity with neural architectures or programming languages. Therefore, this step benefits significantly from a natural language interface. The knowledge declaration stage seeks to construct a concept graph that represents the structure of tasks and incorporates domain-specific logical constraints. A snippet of code in DomiKnowS, illustrating the representation of the Natural Language Inference (NLI) task graph structure and logical constraints, is provided in Figure 1.

3.1 Graph Structure

The graph defines the input-output structure and dependencies of the task, featuring nodes representing concepts and edges denoting relationships. In NLP tasks, input concepts include sentences, paragraphs, phrases, words, and tokens, while vision tasks involve concepts like images and bounding boxes. In structured prediction, task labels are considered output concepts. Each decision concept must be anchored in an input concept; for instance, the sentence_class is connected to the sentence concept, and named entity tags are linked to the phrase concept. DomiKnowS uses 'is_a' for such connections, 'contains' for relationships between a concept and its same-type children, and 'has_a' for many-to-many relationships.

```
with Graph('NLI_Graph') as graph:
    pair = Concept(name='pair')
    premise = Concept(name='premise')
    hypothesis = Concept(name='hypothesis')
    (arg_p, arg_h) = pair.has_a(premise, hypothesis)
    # 3 classes in a multi-class setting
    nli_lb = pair(
        name="nli_class", values=[
            "entailment", "neutral", "contradiction"])
    sym = Concept(name="symmetric")
    s_p1, s_p2 = sym.has_a(arg1=pair, arg2=pair)
    ### Symmetric constraint
    #### Ent(X1, X2) => !CON(X2, X1)
    ifL(andL(nli_lb.entailment('x'),
            existsL(sym('s', path=('x', sym.reversed)))),
        andL(notL(nli_lb.contradiction(
                    path=('x', sym.reversed, s_p2)))))
```

Figure 1. A subset of the concept graph generated for the Natural Language Inference (NLI) task.

3.2 Logical Constraints

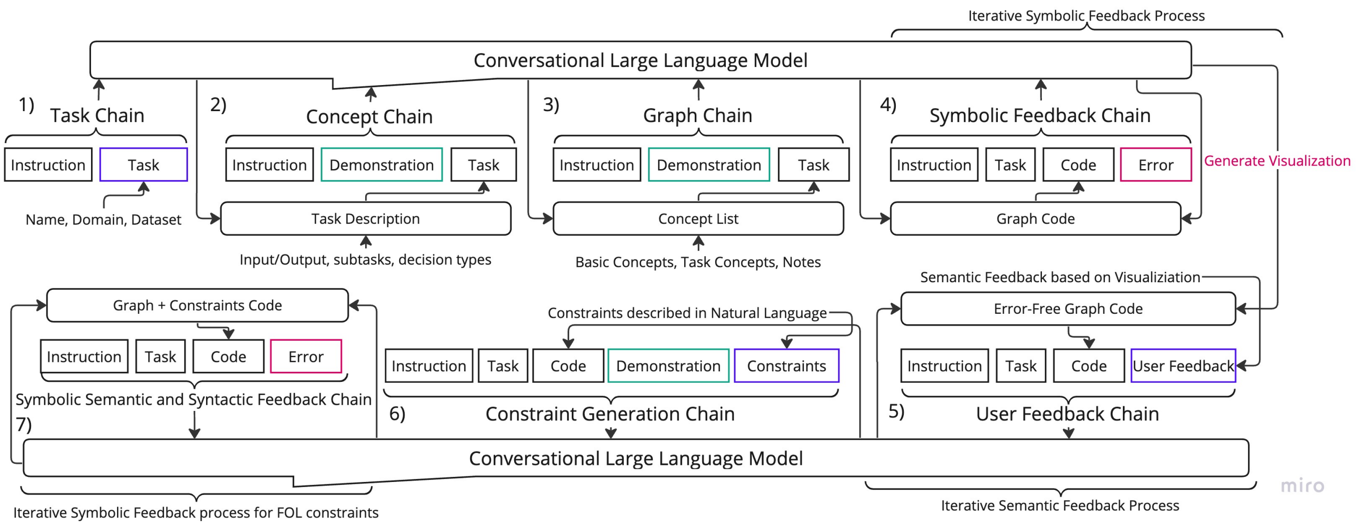
DomiKnowS introduces a specialized Python-based language that employs concepts, edges, variables, and logical operators to declare dependencies in first-order logic between concepts, which is used to define the tasks' domain knowledge in terms of constraints. Significantly, the syntax for specifying first-order logic constraints in DomiKnowS is not observed during the pre-training of large language models, presenting a difficulty in generating outputs that adhere to this format in a zero or few-shot setting.

4 Natural Language to Knowledge Declaration

Our interactive interface employs prompt templates from the LangChain framework[4] and GPT-3.5-turbo[5]. At each stage, user input fills in a predefined template that is consumed by the model to generate the information needed for subsequent steps. We adjust the conversation history to prevent excessive context accumulation, especially where we have an iterative process.

Figure 2 provides an overview of the pipeline. The model may consume a series of underlying natural language instructions, in-context demonstrations, and execution feedback, plus the user input and information from prior interactions at each step. The process begins by gathering basic information, such as the task's name, domain, and dataset name (see step 1 in the figure). Subsequently, the model formulates a clear task description, including the task's input/output and decision types. Users have the flexibility to modify this description. Next, the model compiles a list of concepts (nodes in the graph) representing the task's structure (the input/output concepts, decision

[4] https://www.langchain.com/
[5] https://openai.com/blog/chatgpt/



Iterative Symbolic Feedback Process
Conversational Large Language Model
1) Task Chain
Instruction
Task
Name, Domain, Dataset
2) Concept Chain
Instruction
Demonstration
Task
Task Description
Input/Output, subtasks, decision types
3) Graph Chain
Instruction
Demonstration
Task
Concept List
Basic Concepts, Task Concepts, Notes
4) Symbolic Feedback Chain
Instruction
Task
Code
Error
Graph Code
Generate Visualization
Semantic Feedback based on Visualiziation
Error-Free Graph Code
Instruction
Task
Code
User Feedback
5) User Feedback Chain
Constraints described in Natural Language
Instruction
Task
Code
Demonstration
Constraints
6) Constraint Generation Chain
Graph + Constraints Code
Instruction
Task
Code
Error
Symbolic Semantic and Syntactic Feedback Chain
7)
Conversational Large Language Model
Iterative Symbolic Feedback process for FOL constraints
Iterative Semantic Feedback Process
miro

Figure 2. Overview of the pipeline process. The parts specified with purple, green, and red colors are direct inputs of the user, the (dynamic) in-context examples, and the execution/parser feedback, respectively.

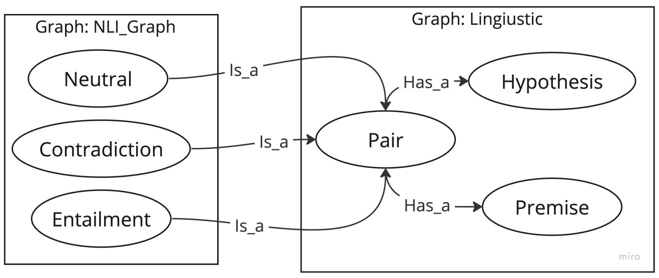


Figure 3. The visualized graph based on the NLI task.

types, and more; See step 2). Users can modify items on this list, which is the foundation for generating an initial concept graph (in Python; See step 3).

The generated code might not be accurate/executable due to the model's lack of familiarity with DomiKnowS' language. Therefore, we implement a loop of symbolic processing of the code and detection of errors while providing the LLM with feedback for refinement. At each step, the model receives errors from the code's parse and execution, which evaluates the graph structure and uses this feedback to refine the code (Step 4). We also seek user feedback once confident in the code's quality. As users need not be familiar with DomiKnowS' notation, they provide feedback based on the visual representation of the code's structure, representing the nodes and edges in the graph (Step 5). Figure 3 visualizes the graph for the NLI task.

Subsequently, users provide the task constraints in natural language. The model aims to translate these expressions into DomiKnowS' Python-based FOL language, which differs somewhat from standard FOL syntax (see step 6). We employ techniques such as dynamic in-context learning, intermediary FOL mapping, and iterative refinement to assist the model in this translation (see step 7).

Here, we briefly discuss some of the important techniques used in the pipeline.

Sampling Strategy We draw multiple samples from the underlying LLM at each step and select the most accurate response through user feedback or automated metrics, such as error counts from the execution. Additionally, if some samples do not have any errors generated by the symbolic processing engine, we prune out the remaining erroneous samples after a certain amount of iterations.

Dynamic In-Context Retrieval Our pipeline involves in-context demonstrations for various tasks in the NLP and vision domains at each step. To enhance model execution speed and minimize noise from irrelevant examples, we selectively include a subset of in-context demonstrations relevant to the target task. We achieve this by creating a vector database of in-context demonstration representations using the OpenAI Embedding service and identifying the most relevant ones for inference based on cosine similarity. This approach reduces computational costs and noise in LLM inference.

Iterative Symbolic Refinement We propose an iterative refinement process in response to LLMs' inability to execute code. Leveraging the representations of structure and constraints in DomiKnowS, we develop syntactic and semantic evaluation parsers to provide feedback. This symbolic parser generates graph structure and constraints, pinpointing semantic and syntactic errors. The parser's feedback serves as instructions for the LLM to refine its initial answer, addressing the challenge of limited prior exposure to DomiKnowS syntax and ensuring accurate responses.

FOL Mapping To enhance constraint generation precision, we utilize an intermediary FOL mapping process. The model translates natural language descriptions into FOL statements and then converts them into DomiKnowS FOL syntax. This approach leverages the model's pre-training exposure to FOL statements. We symbolically extract logical predicates and arguments from the graph to improve prediction by offering additional input. FOL mapping assists LLMs in capturing semantics through FOL representations, enabling users to validate constraints based on these statements, assuming perfect alignment between the two representations.

5 Evaluation and Analysis

We evaluated our natural language interface given various tasks in NLP, Vision, and Constraint Satisfaction Problems (CSP) based on automatic (similarity-based) and human judgments.

5.1 Automatic Evaluation

We devise a set of metrics customized for each step of the process, aligning with the expected output at every stage. This evaluation covers a benchmark of 14 tasks, spanning Entity Mention and Relation Extraction, Hierarchical Classification, Sudoku, Procedural Reasoning, Causal Reasoning, Natural Language Inference, and more. At each stage, we utilize the ground-truth input from the dataset for evaluation. A distinct evaluation approach for end-to-end mode, incorporating human intervention, will be discussed in the next section.

5.1.1 Task, Concept, and Graph Generation

Table 1 reports the results of automatic metrics evaluated on the task description, concepts, and initial and fixed graphs. The outcomes demonstrate the success of our proposed tool in generating the information at each step. The large number of error-free tasks after refinement and the higher performance of the model using sampling indicate the effectiveness of both of these techniques in enhancing the model's performance. Notably, generating the concept list shows a higher error rate, especially in tasks like CIFAR-100 [17], where decision space is large, and the model often falls short by adding 'etc.' to the label set instead of listing all classes. The limited differences in predicted nodes and edges compared to the ground truth suggest that human intervention can effectively address graph issues by instructing the model to add or remove small components. Although dynamic retrieval can help reduce the cost and increase the speed of the process, it slightly hurts the model's performance at these phases.

5.1.2 Constraints Generation

To evaluate constraints, we examine the model's ability to generate semantically and syntactically correct constraints concerning the graph structure and Python syntax within DomiKnowS. The evaluation further details the performance using the intermediary FOL statements and the dynamic retrieval for in-context demonstrations.

Table 2 shows the experiment results for constraint evaluation. Our symbolic parser significantly contributes to resolving errors in the feedback loop, demonstrating the effectiveness of generating well-described errors for output refinement. Experiments indicate that a small sampling factor (3 samples) enhances model accuracy, and FOL mapping serves as a valuable intermediary layer, reducing the need for iterative constraint adjustments. Manual evaluation reveals that while most generated constraints (even in erroneous tasks) are semantically accurate, issues arise in adapting to the DomiKnowS language due to limited knowledge and resources. While dynamic retrieval aids in direct constraint generation, it hinders performance in the FOL setting. This can be attributed to the lack of exposure to mappings from FOL statements into the DomiKnowS language, which tends to encourage the model to employ FOL operations unsupported in DomiKnowS.

5.2 Human Evaluation

The human evaluation assesses two key aspects of our demo. Firstly, it measures the comparative ease and utility of our proposed demo in contrast to the conventional method of navigating the documentation and manually crafting programs within the Python package of the DomiKnowS library. Secondly, the evaluation provides insights into the accuracy of each step in the interactive process, detailing the quantity and nature of user interventions required for satisfactory performance across diverse tasks. Additionally, we aim to assess whether the proposed tool facilitates users unfamiliar with the DomiKnowS programming language in validating the model's responses.

5.2.1 Interface V.S. Coding

We assess user experiences in crafting structured prediction tasks in DomiKnowS with two volunteer groups (5 people per group). Each group received a 20-minute introduction to structured prediction and a high-level overview of DomiKnowS. The first group used the demo interface, while the second group, with access to DomiKnowS documents and examples, manually composed concept graphs and logical constraints. Tasks included hierarchical image classification, sentiment analysis, and entity mention and relation extraction.
Groups' Sub-tasks: Group 1: Describe the task and interact with the interface. Group 2: Read the documentation and examples; Write the code.
Average Time of Implementation: Group 1: 20 minutes, Group 2: 1 hour
Time Splits: Group 1: 60% user input - 40% model output. Group 2: 35% documents - 65% coding
Findings: Human effort using the interface is reduced 5 times. Opting for the local LLM version instead of the API can further accelerate final code generation. Notably, the second group often sought additional guidance on task modeling, while the tools integrated into the demo interface effectively guided the first group. Additionally, we instructed the second group to revisit the task using the demo interface, allocating them a 10-15 minute timeframe. The consensus within this group was that the demo interface provided a significantly superior and more user-friendly experience.

5.2.2 Interactive Setting

To evaluate the demo in an end-to-end interactive setting, we asked two volunteers to implement a total of **seven** tasks while recording their every interaction with the interface.
Duration: The average time for the process was 17 minutes. The most time-consuming part was the constraint generation, taking more time for both the user to write constraints and the model to generate them (averaging 150 seconds for model responses).
Task Descriptions: In 6 out of 7 tasks, the user removed only additional and unnecessary information. In one task, the user extended the decision set by replacing the word 'etc.' with actual labels.
Concept List: In 4 out of 7 tasks, one sample was correct, while in the remaining tasks, the user had to remove or add less than two concepts to the list.
Concept Graph: In 4 out of 7 tasks, the correct graph was generated without user intervention. In 2 out of 7 tasks, one interaction was needed. In 1 out of 7 tasks, the user interacted five times to remove a wrongly included relationship between concepts where multiple relationships existed. The visualization tool notably provided users with all the necessary information to evaluate the graph.
Constraints: Most of the constraints were both semantically and syntactically correct. In cases with erroneous results, the verification process could capture the error but not resolve it correctly. Errors were mainly due to using syntax close to FOL for operations not directly supported in DomiKnowS, like equality between multiple variables. Remarkably, in cases such as constraints in a sudoku table, the model detected similar patterns and used for-loops to implement multiple constraints with similar logical structures but variant logical predicates.

In-Context	# Samples	Task Description (BertScore)	Concept List Diff	Initial Graph		Refined Graph
				Error-Free	Diff	Error-Free
Full	1	73.9%	20.8%	12/14	0.8 N, 3.3 E	14/14
Full	3	**79.3**%	**20.6**%	**13/14**	**0.76 N, 2 E**	14/14
Dynamic	1	73.9%	26.8%	10/14	2.1N, 2.6 E	14/14
Dynamic	3	**79.3**%	26.8%	**13/14**	1.5 N, 2.1 E	14/14

Table 1. Summary of automatic evaluation results for generating task components (description, concept list, initial graph, and post-fix graph). In full in-context, 4 demonstrations are used, while Dynamic has 1. n/N indicates successful graph parsing of n out of N tasks without structural issues. x N, y E represents average x differences in nodes and y differences in edges compared to the ground truth. When using more than 1 sample, the result of the best sample is reported, mimicking human intervention in selecting the optimal answer.

Setting	In-Context	# Samples	Error-Free Tasks	% Err Type in Resolved Err	% Err Type in Unresolved Err
Direct	Full	1	2 D, 6 I	53% P, 47% S	79% P, 21% S
		3	4 D, 9 I	48%P, 52% S	66% P, 34% S
	Dynamic	1	5 D, 5 I	70% P, 30% S	70% P, 30% S
		3	6 D, 7 I	41% P, 59% S	64%P, 36%S
FOL	Full	1	8 D, 0 I	90% P*, 10% S	80% P, 20%S
		3	13 D, 0 I	77%P, 23% S	82%P, 18%S
	Dynamic	1	5 D, 2 I	65%P, 35% S	80% P, 20% S
		3	10 D, 1 I	65%P, 35%S	75%P, 25% S

Table 2. Results for constraint generation and the ratio of error types in resolved/unresolved errors during iterative feedback. There are a total of 14 tasks, with 4 demonstrations in full in-context and 1 in Dynamic. 'i D, j I' denotes i tasks are correct without the need for feedback and j are fixed after the iterative feedback. $i\%$ P, $j\%$ S indicates that $i\%$ of resolved/unresolved errors are Python errors, while $j\%$ are errors caught and reported in a custom prompt with our symbolic parser. In settings with multiple samples, a task is error-free if at least one sample has no errors.

6 Conclusion and Future Work

This demo introduced a novel tool for translating natural language prompts into formal representations of structured prediction tasks and the integration of domain knowledge in learning. Using large language models, we utilize predefined prompt templates, in-context learning, dynamic retrieval, mapping to intermediary representations, sampling, pruning, and iterative symbolic parsing and execution to refine model outputs. Our evaluation, incorporating automatic measurements and human assessments, highlights the tool's ease of use, high accuracy, and the effectiveness of proposed techniques in generating declarative code. Future directions include exploring advancements in retrieving neural models to establish connections between symbolic representations and neural units, facilitating end-to-end neuro-symbolic model creation, composition, and training.

Limitations

Our demo is bound to the limitations of the tools it is built upon. First, our demo can only generate task representations for tasks in the space of structured prediction tasks as supported by the DomiKnowS framework. As such, it might not be perfectly aligned with many generative AI paradigms. Second, our current interface requires access to the OpenAI API and GPT-4/GPT-3.5. However, developing the system utilizing Langchain would allow us to switch the underlying LLM to open-source and local models easily.

References

[1] K. Ahmed, T. Li, T. Ton, Q. Guo, K.-W. Chang, P. Kordjamshidi, V. Srikumar, G. Van den Broeck, and S. Singh. Pylon: A pytorch framework for learning with constraints. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 319–324. PMLR, 2022.

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[6] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[7] H. R. Faghihi, Q. Guo, A. Uszok, A. Nafar, and P. Kordjamshidi. Domiknows: A library for integration of symbolic domain knowledge in deep learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 231–241, 2021.

[8] M. Freitag and Y. Al-Onaizan. Beam search strategies for neural machine translation. *ACL 2017*, page 56, 2017.

[9] Q. Guo, H. R. Faghihi, Y. Zhang, A. Uszok, and P. Kordjamshidi. Inference-masked loss for deep structured output learning. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 2754–2761, 2021.

[10] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, L. Benson, L. Sun, E. Zubova, Y. Qiao, M. Burtell, et al. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*, 2022.

[11] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. In *54th ACL*, pages 2410–2420, 2016.

[12] J. Huang, Z. Li, B. Chen, K. Samel, M. Naik, L. Song, and X. Si. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. *Advances in Neural Information Processing Systems*, 34: 25134–25145, 2021.

[13] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

[14] J. Kocoń, I. Cichecki, O. Kaszyca, M. Kochanek, D. Szydło, J. Baran, J. Bielaniewicz, M. Gruza, A. Janz, K. Kanclerz, et al. Chatgpt: Jack of all trades, master of none. *Information Fusion*, page 101861, 2023.

[15] P. Kordjamshidi, D. Roth, and K. Kersting. Systems ai: A declarative learning based programming perspective. In *International Joint Conference on Artificial Intelligence*, 2018. URL https://api.semanticscholar.org/CorpusID:44080347.

[16] P. Kordjamshidi, D. Roth, and K. Kersting. Declarative learning-based programming as an interface to ai systems. *Frontiers in artificial intel-*

ligence, 5:755361, 2022.

[17] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

[18] J. Lloyd. Practical advantages of declarative programming. In *Unknown*, pages 3 – 17, 1994. Conference Proceedings/Title of Journal: Joint Conference on Declarative Programming.

[19] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

[20] Y. Nandwani, A. Pathak, P. Singla, et al. A primal dual formulation for deep learning with constraints. In *Advances in Neural Information Processing Systems*, pages 12157–12168, 2019.

[21] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. Wang, and X. V. Lin. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128. PMLR, 2023.

[22] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[23] L. Pan, A. Albalak, X. Wang, and W. Y. Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023.

[24] G. Poesia, K. Gandhi, E. Zelikman, and N. D. Goodman. Certified reasoning with language models. *arXiv preprint arXiv:2306.04031*, 2023.

[25] H. Rajaby Faghihi, A. Nafar, C. Zheng, R. Mirzaee, Y. Zhang, A. Uszok, A. Wan, T. Premsri, D. Roth, and P. Kordjamshidi. Gluecons: A generic benchmark for learning under constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9552–9561, Jun. 2023. doi: 10.1609/aaai.v37i8.26143.

[26] D. Roth. Learning Based Programming. Technical Report UIUCDCS-R-99-2127, 10 1999. URL http://cogcomp.org/papers/Roth99c.pdf.

[27] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[28] T. Scholak, N. Schucher, and D. Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *EMNLP*, pages 9895–9901, 2021.

[29] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang. Hugging-gpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[30] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[31] V. Viswanathan, C. Zhao, A. Bertsch, T. Wu, and G. Neubig. Prompt2model: Generating deployable models from natural language instructions. *arXiv preprint arXiv:2308.12261*, 2023.

[32] Z. Wu, L. Zhang, C. Cao, X. Yu, H. Dai, C. Ma, Z. Liu, L. Zhao, G. Li, W. Liu, et al. Exploring the trade-offs: Unified large language models vs local fine-tuned models for highly-specific radiology nli task. *arXiv preprint arXiv:2304.09138*, 2023.

[33] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, pages 5502–5511. PMLR, 2018.

[34] Z. Yang, A. Ishay, and J. Lee. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*, 2023.

[35] P. Yu and S. Bach. Alfred: A system for prompted weak supervision. *arXiv preprint arXiv:2305.18623*, 2023.

[36] T. Zhang, T. Yu, T. Hashimoto, M. Lewis, W.-t. Yih, D. Fried, and S. Wang. Coder reviewer reranking for code generation. In *International Conference on Machine Learning*, pages 41832–41846. PMLR, 2023.

[37] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.