Neurosymbolic Programming in Scallop: Principles and Practice

Other titles in Foundations and Trends® in Programming Languages

From Fine- to Coarse-Grained Dynamic Information Flow Control and Back
Marco Vassena, Alejandro Russo, Deepak Garg, Vineet Rajani and Deian Stefan
ISBN: 978-1-63828-218-1

Probabilistic Trace and Testing Semantics: The Importance of Being Coherent
Marco Bernardo
ISBN: 978-1-63828-074-3

Neurosymbolic Programming
Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama and Yisong Yue
ISBN: 978-1-68083-934-0

Introduction to Neural Network Verification
Aws Albarghouthi
ISBN: 978-1-68083-910-4

Refinement Types: A Tutorial
Ranjit Jhala and Niki Vazou
ISBN: 978-1-68083-884-8

Shape Analysis
Bor-Yuh Evan Chang, Cezara Drăgoi, Roman Manevich, Noam Rinetzky and Xavier Rival
ISBN: 978-1-68083-732-2

Neurosymbolic Programming in Scallop: Principles and Practice

Ziyang Li
University of Pennsylvania
liby99@seas.upenn.edu

Jiani Huang
University of Pennsylvania
jianih@seas.upenn.edu

Jason Liu
University of Pennsylvania
jasonhl@seas.upenn.edu

Mayur Naik
University of Pennsylvania
mhnaik@seas.upenn.edu

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

Z. Li *et al.*. *Neurosymbolic Programming in Scallop: Principles and Practice*. Foundations and Trends® in Programming Languages, vol. 8, no. 2, pp. 118–249, 2024.

ISBN: 978-1-63828-485-7

Foundations and Trends® in Programming Languages

Volume 8, Issue 2, 2024

Editorial Scope

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract Interpretation
- Compilation and Interpretation Techniques
- Domain Specific Languages
- Formal Semantics, including Lambda Calculi, Process Calculi, and Process Algebra
- Language Paradigms
- Mechanical Proof Checking
- Memory Management
- Partial Evaluation
- Program Logic
- Programming Language Implementation
- Programming Language Security
- Programming Languages for Concurrency
- Programming Languages for Parallelism
- Program Synthesis
- Program Transformations and Optimizations
- Program Verification
- Runtime Techniques for Programming Languages
- Software Model Checking
- Static and Dynamic Program Analysis
- Type Theory and Type Systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2024, Volume 8, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Contents

Neurosymbolic Programming in Scallop: Principles and Practice

Ziyang Li, Jiani Huang, Jason Liu and Mayur Naik

University of Pennsylvania, USA; liby99@seas.upenn.edu, jianih@seas.upenn.edu, jasonhl@seas.upenn.edu, mhnaik@seas.upenn.edu

ABSTRACT

Neurosymbolic programming combines the otherwise complementary worlds of deep learning and symbolic reasoning. It thereby enables more accurate, interpretable, and domain-aware solutions to AI tasks. We introduce Scallop, a general-purpose language and compiler toolchain for developing neurosymbolic applications. A Scallop program specifies a suitable decomposition of an AI task's computation into separate learning and reasoning modules. Learning modules are built using existing machine learning frameworks and range from custom neural models to foundation models for language, vision, and multi-modal data. Reasoning modules are specified in a declarative logic programming language based on Datalog which supports expressive features such as recursion, aggregation, negation, and probabilistic programming over structured relations.

Scallop's compiler enables to automatically train neurosymbolic programs in a data- and compute-efficient manner using an end-to-end differentiable reasoning framework. Scallop also supports features useful for building real-world applications such as user-defined data types, and foreign interfaces.

Ziyang Li, Jiani Huang, Jason Liu and Mayur Naik (2024), "Neurosymbolic Programming in Scallop: Principles and Practice", Foundations and Trends® in Programming Languages: Vol. 8, No. 2, pp 118–249. DOI: 10.1561/2500000059.

We demonstrate programming in Scallop for applications that span the domains of image and video processing, natural language processing, planning, and information retrieval in a variety of learning settings such as supervised learning, reinforcement learning, rule learning, contrastive learning, and in-context learning.

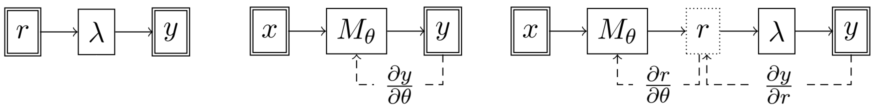
1

Introduction

1.1 Neurosymbolic Programming

Classical algorithms and deep learning embody two prevalent paradigms of modern programming. Classical algorithms are well suited for exactly-defined tasks, such as sorting a list of numbers or finding a shortest path in a graph. Deep learning, on the other hand, is well suited for tasks that are not tractable or feasible to perform procedurally, such as detecting objects in an image or parsing natural language text. These tasks are typically specified using a set of input-output training data, and solving them involves learning the parameters of a deep neural network to fit the data using gradient-based methods.

The two paradigms are complementary in nature. For instance, a classical algorithm such as the logic program λ shown in Figure 1.1a is interpretable but operates on limited, structured input r. On the other hand, a deep neural network such as M_θ shown in Figure 1.1b can operate on rich, unstructured input x but is not interpretable. Modern applications demand the capabilities of both paradigms. Examples include question answering (Rajpurkar *et al.*, 2016), code completion (Chen *et al.*, 2021), and mathematical problem solving (Lewkowycz *et al.*, 2022), among many others. For instance, code completion requires



(a) Logic program. **(b)** Neural model. **(c)** A basic neurosymbolic program.

Figure 1.1: Comparison of different paradigms. Logic program λ accepts only structured input r whereas neural model M_θ with parameter θ can operate on unstructured input x. Supervision is provided on data indicated in double boxes. Under *algorithmic supervision*, a neurosymbolic program must learn θ without supervision on r.

deep learning to comprehend programmer intent from the code context, and classical algorithms to ensure that the generated code is correct. A natural and fundamental question then is how to program such applications by integrating the two paradigms.

Neurosymbolic programming is an emerging paradigm that aims to fulfill this goal (Chaudhuri *et al.*, 2021). It seeks to integrate symbolic knowledge and reasoning with neural architectures for better efficiency, interpretability, and generalizability than the neural or symbolic counterparts alone. Consider the task of handwritten formula evaluation (Li *et al.*, 2020), which takes as input a formula as an image, and outputs a number corresponding to the result of evaluating it. An input-output example for this task is $\langle x =$ 1+3÷5$, y = 1.6\rangle$. A neurosymbolic program for this task, such as the one shown in Figure 1.1c, might first apply a convolutional neural network M_θ to the input image to obtain a structured intermediate form r as a sequence of symbols [‘1’, ‘+’, ‘3’, ‘/’, ‘5’], followed by a classical algorithm λ to parse the sequence, evaluate the parsed formula, and output the final result 1.6.

Despite significant strides in individual neurosymbolic applications (Yi *et al.*, 2018; Mao *et al.*, 2019; Chen *et al.*, 2020; Li *et al.*, 2020; Minervini *et al.*, 2020a; Wang *et al.*, 2019), there is a lack of a language with compiler support to make the benefits of the neurosymbolic paradigm more widely accessible. We set out to develop such a language and identified five key criteria that it should satisfy in order to be practical. These criteria, annotated by the components of the neurosymbolic program in Figure 1.1c, are as follows:

1. A symbolic data representation for r that supports diverse kinds of data, such as image, video, natural language text, tabular data, and their combinations.

2. A symbolic reasoning language for λ that expresses common reasoning patterns such as recursion, negation, and aggregation.

3. An automatic and efficient differentiable reasoning engine for learning ($\frac{\partial y}{\partial r}$) under *algorithmic supervision*, i.e., supervision on observable input-output data (x, y) but not r.

4. The ability to tailor learning ($\frac{\partial y}{\partial r}$) to individual applications' characteristics, since non-continuous loss landscapes of symbolic programs hinder learning using a one-size-fits-all method.

5. A mechanism to leverage and integrate with existing training pipelines ($\frac{\partial r}{\partial \theta}$), implementations of neural architectures and models M_θ, and hardware (e.g. GPU) optimizations.

1.2 Scallop: What and Why

We have developed Scallop, a programming language that realizes all of the above criteria. The key insight underlying Scallop is its choice of three inter-dependent design decisions: a relational model for symbolic data representation, a declarative language for symbolic reasoning, and a provenance framework for differentiable reasoning.

Our design choices were inspired by the following key observations. First, much of the world's data is stored in relational databases. Relations are also flexible enough to represent diverse kinds of data ranging from high-level visual and language features, to formal programs, to molecular structures. Second, a declarative language for symbolic reasoning allows computation to be expressed concisely via high-level rules, thereby alleviating programmer effort. Finally, the relational paradigm offers a suitable abstraction for various advanced features needed for neurosymbolic programming, such as query planning, hardware acceleration, and probabilistic and differentiable reasoning.

Our aim with Scallop is to provide a cohesive language and framework for integrating neural and symbolic components. In doing so, we seek to enable programmers to build neurosymbolic solutions that are more efficient, generalizable, and interpretable.

1.3 Building Blocks of Neurosymbolic Solutions

A language that integrates neural and symbolic components can be applied to construct diverse and adaptable solutions. Broadly, a neurosymbolic solution to any given task involves the flexible interplay of neural and symbolic components, each serving distinct yet complementary roles in problem-solving. From the existing literature, several building blocks have emerged as crucial for effective neurosymbolic solutions, as depicted in Figure 1.2. We proceed to discuss each of these core building blocks in detail.

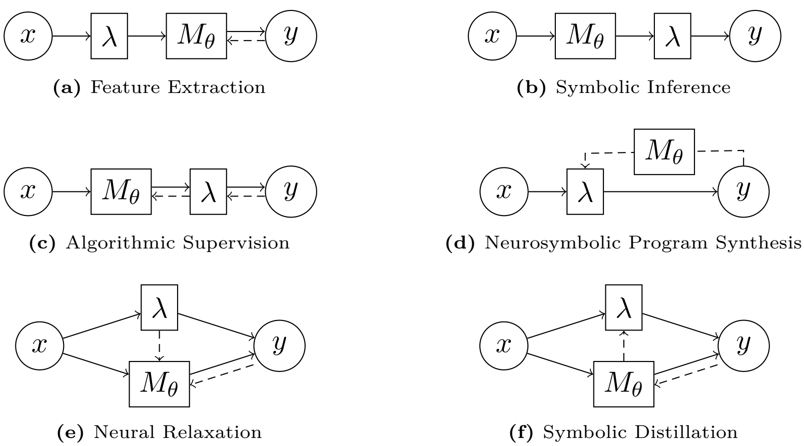
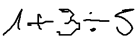


Figure 1.2: Neurosymbolic compositions of neural component (M_θ) and symbolic component (λ), which serve as building-blocks for more complex neurosymbolic applications. We use solid arrows to denote forward data-flows, and dashed arrows to denote backward data-flows used to supervise the learning of the target component.

Feature Extraction The feature extraction process involves deriving symbolic features from an input x through a symbolic component, denoted here as λ, before passing these features to a neural model M_θ for training. Although feature extraction is an established practice in machine learning and typically not classified as neurosymbolic, it nevertheless exemplifies a functional integration of symbolic and neural elements. In this approach, learning is confined to the neural component, while the symbolic aspect serves to pre-process the input data.

Notably, advanced feature extraction goes beyond simple tabular data and often incorporates sophisticated reasoning mechanisms to construct complex data structures. For instance, in program analysis, source code can be pre-processed into intricate structures such as abstract syntax trees (ASTs), data-flow graphs, symbolic constraints, or relational databases (Dinella *et al.*, 2020; Li *et al.*, 2021; Zhu *et al.*, 2024). Neural networks may thus benefit from more comprehensive, structured information for downstream tasks, such as proposing bug fixes, detecting vulnerabilities, and analyzing type information even within binary code.

Symbolic Inference Symbolic inference involves performing posterior analysis on the outputs of a neural network M_θ using a symbolic component λ provided by a programmer. This analysis can serve various purposes, such as filtering nonsensical outputs, verifying output integrity, or combining multiple information sources symbolically to derive additional insights. Though straightforward in concept, an advanced symbolic inference component λ may handle probabilistic information, deriving a distribution rather than just the most likely output.

For instance, in the task of handwritten formula recognition $\langle x =$ $, y = 1.6\rangle$, after the neural network generates probability distributions for individual symbols, a probabilistic symbolic inference engine could synthesize a distribution over possible rational numbers. Another example is RNA secondary structure prediction, where a neural network predicts per-nucleotide structures, and a probabilistic RNA folding algorithm then parses this probabilistic sequence to generate the top-k most likely structural parses. In Section 5, we cover many symbolic inference solutions where the M_θ are foundation models.

Algorithmic Supervision Algorithmic supervision extends symbolic inference by enabling the symbolic component λ to propagate learning signals to the neural network M_θ. As before, we assume that λ is provided by the programmer. While Figure 1.1 demonstrates one example of algorithmic supervision through differentiability in λ, it generally suffices for λ to propagate the learning signal. In this way, the symbolic "algorithm" λ serves as a guiding supervisor for the neural network M_θ.

Algorithmic supervision also functions as a form of weak supervision, as it does not require direct, fully supervised labels for M_θ; only the end label y is needed. This reduces the need for extensive data labeling or feature engineering, simplifying the training process. Numerous applications in Scallop leverage this approach, including the previously mentioned task of learning to evaluate handwritten formulas (Li *et al.*, 2020; Li *et al.*, 2023). This tutorial explores additional, advanced examples of algorithmic weak supervision in Section 6.

Neurosymbolic Program Synthesis Neurosymbolic program synthesis involves learning the symbolic program λ with the support of neural networks. This paradigm resembles the classical syntax-guided synthesis task (Alur *et al.*, 2013), but replaces the traditional algorithmic synthesis procedure with a neural network M_θ. Here, the symbolic program λ is responsible for generating the expected outputs, and it may be iteratively refined to better align with a dataset.

This approach offers the advantage of interpretability, as the learned symbolic component is a white-box program that can be inspected and verified by humans (Ellis *et al.*, 2022). Traditionally, synthesizing λ requires defining a limited domain-specific language (Ellis *et al.*, 2020) since general-purpose languages render synthesis computationally intractable. However, with the recent development of large language models (LLMs) capable of generating programs in general-purpose languages like Python, the synthesis of λ can now be achieved more efficiently (Ma *et al.*, 2024).

Neural Relaxation Neural relaxation involves relaxing a deterministic and discrete symbolic reasoning component λ by replacing certain components in the pipeline with neural networks M_θ. This enables portions of previously symbolic components to be approximated by neural networks, improving adaptability to unseen scenarios.

For instance, consider the challenge of designing a neurosymbolic controller for drones: while effective deterministic controllers exist for standard maneuvers, they may struggle to adapt to out-of-domain scenarios, such as operating near the ground, in strong winds, or in proximity to other drones. By relaxing certain aspects of the controller into a neural network M_θ, the system gains greater flexibility and responsiveness in handling such scenarios, while being able to learn rapidly (O'Connell *et al.*, 2022; Csomay-Shanklin *et al.*, 2024).

Symbolic Distillation Symbolic distillation extracts information from a black-box neural network and converts it into a symbolic form λ. Although this process involves generating and refining λ, similar to neurosymbolic program synthesis, symbolic distillation focuses on translating otherwise uninterpretable weights from a well-trained neural network M_θ into an interpretable form.

This technique has been applied to scientific discovery in fields such as animal behavior analysis (Sun *et al.*, 2022). A symbolic program describing behaviors like "two mice running towards each other" can be distilled from a neural network trained on data of mice interactions. Another application is explanation synthesis for predicting cancer patient mortality (Wu *et al.*, 2024). For a model trained to predict 6-month mortality, symbolic distillation can generate explanations of specific predictions, providing clearer insights for clinical decision-making supported by machine learning systems.

Other Compositions In addition to the primary building blocks, there are other notable neurosymbolic compositions. For example, AlphaGo (Silver *et al.*, 2016) is centered around a symbolic algorithm—Monte Carlo Tree Search—with neural networks for policy evaluation and move selection, creating a synergistic decision-making process. On the other hand, ChatGPT plugins (OpenAI, 2023a) use a large language model as the primary system, which can invoke symbolic components like a Python interpreter, database retrieval, or web search to enhance functionality. As the field of neurosymbolic AI continues to evolve, we anticipate that more diverse and innovative compositions will emerge, broadening the scope and applications of neurosymbolic approaches.

1.4 Application Domains

In this section, we discuss the data modalities for which Scallop is best suited and explore the application domains where Scallop has shown effectiveness. We also identify the limitations of Scallop, highlighting tasks where it may be less effective.

Scallop can be broadly applied to applications that require both neural models and programmatic reasoning modules. It is particularly useful when the neural model requires additional training. With a fully differentiable, end-to-end neurosymbolic pipeline, strong supervision is not necessary for the neural model. Instead, *algorithmic supervision* can be used, offering benefits such as data efficiency and generalizability.

Data Modalities Scallop is capable of handling diverse data modalities by virtue of being based on the relational data model. The relational paradigm enables it to work seamlessly with existing relational databases and tabular data, encompassing information from knowledge bases, electronic health records, and internet documents. Additionally, natural language data from NLP tasks can be ingested in various forms: as raw sentences, embeddings (tensors), or structured representations such as relational databases or functional programs. Image data from computer vision can be converted into semantic representations like scene graphs. Videos, which extend images with a temporal dimension, can similarly be represented as spatio-temporal scene graphs for analysis in Scallop. Computer programs can be transformed into relational databases, capturing detailed information such as abstract syntax trees and control-flow graphs.

Application Domains We have applied Scallop across diverse domains, including natural language processing (NLP), computer vision (CV), planning, program and security analysis, bioinformatics, and healthcare. In the domain of NLP, we have applied Scallop to tasks that require reasoning, such as retrieving documents in a database, or analyzing data from sources such as electronic health records or legal documents. In the domain of computer vision, rather than focusing on low-level perception tasks like object segmentation and tracking, we have applied Scallop to hybrid tasks such as visual question answering and for supporting the training of scene graph generation models. In security analysis, we have applied Scallop to tasks like taint analysis, vulnerability detection, and fault localization. In bioinformatics, we have employed Scallop in applications such as predicting RNA secondary structures and RNA splicing. It is important to note that not all Scallop solutions follow a uniform architecture. We adapt different building blocks (Figure 1.2) depending upon each task's unique characteristics.

Applications Where Scallop May Be Less Effective We identify three examples where Scallop may not significantly enhance the task-solving process due to challenges in defining the reasoning component or the appropriate intermediate representation.

1. *Generating Text with Subjective Criteria*. A common use-case of language models like GPT is generating text that satisfies subjective criteria in style or content, such as empathy or political neutrality. While language models can generate coherent paragraphs, identifying specific logical components for integration is challenging. The abstract nature of such tasks makes it difficult to pinpoint areas where logical reasoning would offer substantial value beyond what current language models provide.

2. *Basic Math Calculations* (e.g., $+$, $-$, $\times$, $\div$). This task is inherently symbolic and straightforward. Existing tools like Python or MATLAB can perform these operations directly, and there is no clear need for a perceptual model. The task is purely logical and lacks components that would benefit from Scallop's relational or perceptual capabilities.

3. *Low-Level Motor Control for Robots*. Scallop's syntax is more suited to defining high-level discrete logical rules rather than handling low-level numerical processing of sensory signals. Thus, for tasks like motor control based on raw sensor inputs, imperative languages such as C or Python may be more effective for specifying the numerical algorithms.

1.5 Intended Audience

Scallop is built on the logic programming paradigm and integrates seamlessly with machine learning frameworks like PyTorch through Python bindings. As such, we assume readers are familiar with foundational concepts in logic, machine learning, basic calculus (specifically differentiation), and the Python programming language. This tutorial covers topics including programming language syntax and semantics, probabilistic theories and approximations, and the design and implementation of machine learning systems. While it also explores applications in natural language processing and computer vision, we provide accessible introductions to each task. Overall, this tutorial is designed for readers seeking a practical, foundational understanding of neurosymbolic programming with Scallop, covering both theoretical concepts and real-world applications.

1.6 Outline

We cover the core Scallop language in Section 2 starting from the basics of relational programming. We then describe our core reasoning module in Section 3 which dives deeper into the internals of Scallop and our provenance framework. We show the core programming constructs in Scallop that allow for scalable differentiable reasoning. Next, Section 4 presents a few motivating tasks showcasing Scallop's ability to concisely and effectively define neurosymbolic applications. Section 5 connects Scallop with foundation models. We present a few more advanced neurosymbolic applications in Section 6. Finally, Section 7 concludes with a discussion of limitations and future directions.

References

Abiteboul, S., R. Hull, and V. Vianu. (1995). *Foundations of Databases: The Logical Level.* Addison-Wesley Longman Publishing Co., Inc.

Albers, S., A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas. (2009). *Automata, Languages and Programming: 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I.* Vol. 5555. Springer Science & Business Media.

Alur, R., R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. (2013). "Syntax-guided synthesis". *2013 Formal Methods in Computer-Aided Design*: 1–8. URL: https://api.semanticscholar.org/CorpusID:6705760.

Beurer-Kellner, L., M. Fischer, and M. Vechev. (2022). "Prompting Is Programming: A Query Language For Large Language Models". In: *PLDI.*

Bommasani, R., D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, and et al. (2021). "On the Opportunities and Risks of Foundation Models". arXiv: 2108.07258.

Bubeck, S., V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, *et al.* (2023). "Sparks of artificial general intelligence: Early experiments with GPT-4". arXiv: 2303.12712.

Chaki, S., E. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. (2005). "Concurrent software verification with states, events, and deadlocks". *Formal Aspects of Computing*. 17(4): 461–483.

Chang, C.-Y., D.-A. Huang, Y. Sui, L. Fei-Fei, and J. C. Niebles. (2019). "D3tw: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3546–3555.

Chaudhuri, S., K. Ellis, O. Polozov, R. Singh, A. Solar-Lezama, Y. Yue, *et al.* (2021). "Neurosymbolic Programming". *Foundations and Trends in Programming Languages*. 7(3). DOI: 10.1561/2500000049.

Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.* (2021). "Evaluating Large Language Models Trained on Code". arXiv: 2107.03374.

Chen, X., C. Liang, A. W. Yu, D. Zhou, D. Song, and Q. V. Le. (2020). "Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension". In: *International Conference on Learning Representations (ICLR)*.

Cho, K., B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *EMNLP*.

Cong, Y., W. Liao, H. Ackermann, M. Y. Yang, and B. Rosenhahn. (2021). "Spatial-Temporal Transformer for Dynamic Scene Graph Generation". *CoRR*. abs/2107.12309. URL: https://arxiv.org/abs/2107.12309.

Csomay-Shanklin, N., W. D. Compton, I. D. J. Rodriguez, E. R. Ambrose, Y. Yue, and A. D. Ames. (2024). "Robust Agility via Learned Zero Dynamics Policies". URL: https://arxiv.org/abs/2409.06125.

Damen, D., H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. (2018). "Scaling Egocentric Vision: The EPIC-KITCHENS Dataset". In: *European Conference on Computer Vision (ECCV)*.

Damen, D., H. Doughty, G. M. Farinella, A. Furnari, J. Ma, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. (2022). "Rescaling Egocentric Vision: Collection, Pipeline and Challenges for EPIC-KITCHENS-100". *International Journal of Computer Vision (IJCV)*. 130: 33–55. URL: https://doi.org/10.1007/s11263-021-01531-2.

Dannert, K. M., E. Grädel, M. Naaf, and V. Tannen. (2021). "Semiring Provenance for Fixed-Point Logic". In: *Conference on Computer Science Logic (CSL)*. DOI: 10.4230/LIPIcs.CSL.2021.17.

Darwiche, A. (2011). "SDD: A New Canonical Representation of Propositional Knowledge Bases". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. DOI: 10.5591/978-1-57735-516-8/IJCAI11-143.

De Giacomo, G. and M. Y. Vardi. (2013). "Linear temporal logic and linear dynamic logic on finite traces". In: *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. Association for Computing Machinery. 854–860.

Dinella, E., H. Dai, Z. Li, M. Naik, L. Song, and K. Wang. (2020). "HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS".

Ding, X., S. L. Smith, C. Belta, and D. Rus. (2014). "Optimal control of Markov decision processes with linear temporal logic constraints". *IEEE Transactions on Automatic Control*. 59(5): 1244–1257.

Ellis, K., A. Albright, A. Solar-Lezama, J. B. Tenenbaum, and T. J. O'Donnell. (2022). "Synthesizing theories of human language with Bayesian program induction". *Nature Communications*. 13. URL: https://api.semanticscholar.org/CorpusID:251951680.

Ellis, K., C. Wong, M. I. Nye, M. Sablé-Meyer, L. Cary, L. Morales, L. B. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum. (2020). "DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning". *CoRR*. abs/2006.08381. URL: https://arxiv.org/abs/2006.08381.

Fu, T.-J., L. Li, Z. Gan, K. Lin, W. Y. Wang, L. Wang, and Z. Liu. (2021). "Violet: End-to-end video-language transformers with masked visual-token modeling". *arXiv preprint arXiv:2111.12681*.

Gao, L., A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. (2023). "PAL: Program-aided Language Models". arXiv: 2211.10435 [cs.CL].

Grauman, K., A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, M. Martin, T. Nagarajan, I. Radosavovic, S. K. Ramakrishnan, F. Ryan, J. Sharma, M. Wray, M. Xu, E. Z. Xu, C. Zhao, S. Bansal, D. Batra, V. Cartillier, S. Crane, T. Do, M. Doulaty, A. Erapalli, C. Feichtenhofer, A. Fragomeni, Q. Fu, C. Fuegen, A. Gebreselasie, C. González, J. Hillis, X. Huang, Y. Huang, W. Jia, W. Khoo, J. Kolár, S. Kottur, A. Kumar, F. Landini, C. Li, Y. Li, Z. Li, K. Mangalam, R. Modhugu, J. Munro, T. Murrell, T. Nishiyasu, W. Price, P. R. Puentes, M. Ramazanova, L. Sari, K. Somasundaram, A. Southerland, Y. Sugano, R. Tao, M. Vo, Y. Wang, X. Wu, T. Yagi, Y. Zhu, P. Arbeláez, D. Crandall, D. Damen, G. M. Farinella, B. Ghanem, V. K. Ithapu, C. V. Jawahar, H. Joo, K. Kitani, H. Li, R. A. Newcombe, A. Oliva, H. S. Park, J. M. Rehg, Y. Sato, J. Shi, M. Z. Shou, A. Torralba, L. Torresani, M. Yan, and J. Malik. (2021). "Ego4D: Around the World in 3, 000 Hours of Egocentric Video". *CoRR*. abs/2110.07058. URL: https://arxiv.org/abs/2110.07058.

Green, T. J., G. Karvounarakis, and V. Tannen. (2007). "Provenance Semirings". In: *ACM Symposium on Principles of Database Systems (PODS)*. DOI: 10.1145/1265530.1265535.

Gupta, T. and A. Kembhavi. (2022). "Visual Programming: Compositional visual reasoning without training". arXiv: 2211.11559.

Hochreiter, S. and J. Schmidhuber. (1997). "Long short-term memory". *Neural computation*.

Huang, J., Z. Li, B. Chen, K. Samel, M. Naik, L. Song, and X. Si. (2021). "Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning". In: *Conference on Neural Information Processing Systems (NeurIPS)*.

Hudson, D. A. and C. D. Manning. (2018). "Compositional Attention Networks for Machine Reasoning". In: *ICLR*.

Johnson, J., B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick. (2016). "CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning". URL: http://arxiv.org/abs/1612.06890.

Kenton, J. D. M.-W. C. and L. K. Toutanova. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL*.

Kesten, Y., A. Pnueli, and L.-o. Raviv. (1998). "Algorithmic verification of linear temporal logic specifications". In: *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*. Springer. 1–16.

Kim, W., B. Son, and I. Kim. (2021). "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision". arXiv: 2102.03334 [stat.ML].

Kirillov, A., E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.* (2023). "Segment Anything". arXiv: 2304.02643.

Kojima, T., S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. (2022). "Large language models are zero-shot reasoners". In: *NeurIPS*.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. (1998). "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE*. 86(11). DOI: 10.1109/5.726791.

Lewkowycz, A., A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, *et al.* (2022). "Solving Quantitative Reasoning Problems with Language Models".

Li, J., Y. Wang, C. Wang, Y. Tai, J. Qian, J. Yang, C. Wang, J. Li, and F. Huang. (2018). "DSFD: Dual Shot Face Detector". URL: http://arxiv.org/abs/1810.10220.

Li, Q., S. Huang, Y. Hong, Y. Chen, Y. N. Wu, and S.-C. Zhu. (2020). "Closed Loop Neural-Symbolic Learning via Integrating Neural Perception, Grammar Parsing, and Symbolic Reasoning". In: *ICML*. DOI: 10.48550/arXiv.2006.06649.

Li, X.-Y., W.-J. Lei, and Y.-B. Yang. (2022). "From Easy to Hard: Two-stage Selector and Reader for Multi-hop Question Answering". arXiv: 2205.11729 [cs.CL].

Li, Z., J. Huang, and M. Naik. (2023). "Scallop: A Language for Neurosymbolic Programming". In: *PLDI*. DOI: 10.1145/3591280.

Li, Z., A. Machiry, B. Chen, M. Naik, K. Wang, and L. Song. (2021). "ARBITRAR: User-Guided API Misuse Detection". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 1400–1415. DOI: 10.1109/SP40001.2021.00090.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach". arXiv: 1907.11692.

Ma, Y. J., W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. (2024). "Eureka: Human-Level Reward Design via Coding Large Language Models". URL: https://arxiv.org/abs/2310.12931.

Manhaeve, R., S. Dumancic, A. Kimmig, T. Demeester, and L. D. Raedt. (2021). "Neural Probabilistic Logic Programming in DeepProbLog". *Artificial Intelligence*. 298. DOI: 10.1016/j.artint.2021.103504.

Mao, J., C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. (2019). "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision". arXiv: 1904.12584.

McKenna, N., T. Li, L. Cheng, M. J. Hosseini, M. Johnson, and M. Steedman. (2023). "Sources of Hallucination by Large Language Models on Inference Tasks". arXiv: 2305.14552.

Minderer, M., A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, X. Wang, X. Zhai, T. Kipf, and N. Houlsby. (2022). "Simple Open-Vocabulary Object Detection with Vision Transformers". arXiv: 2205.06230 [cs.CV].

Minervini, P., S. Riedel, P. Stenetorp, E. Grefenstette, and T. Rocktäschel. (2020a). "Learning Reasoning Strategies in End-to-End Differentiable Proving". In: *ICML*. arXiv: 2007.06477.

Minervini, P., S. Riedel, P. Stenetorp, E. Grefenstette, and T. Rocktäschel. (2020b). "Learning reasoning strategies in end-to-end differentiable proving". In: *ICML*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.* (2015). "Human-level Control Through Deep Reinforcement Learning". *Nature*. 518(7540). DOI: 10.1038/nature14236.

Nag, S., K. Min, S. Tripathi, and A. K. R. Chowdhury. (2023). "Unbiased Scene Graph Generation in Videos". arXiv: 2304.00733 [cs.CV].

Nakano, R., J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman. (2021). "WebGPT: Browser-assisted question-answering with human feedback". URL: https://arxiv.org/abs/2112.09332.

Nogueira, R. and K. Cho. (2019). "Passage Re-ranking with BERT". arXiv: 1901.04085.

O'Connell, M., G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. (2022). "Neural-Fly enables rapid learning for agile flight in strong winds". *Science Robotics*. 7(66). DOI: 10.1126/scirobotics.abm6597.

OpenAI. (2023a). "ChatGPT Plugins". URL: https://openai.com/index/chatgpt-plugins/.

OpenAI. (2023b). "GPT-4 Technical Report". arXiv: 2303.08774 [cs.CL].

OpenAI *et al.* (2024). "GPT-4 Technical Report". arXiv: 2303.08774 [cs.CL].

Petersen, F. (2022). "Learning with Differentiable Algorithms". *arXiv preprint arXiv:2209.00616*.

Pnueli, A. (1977). "The temporal logic of programs". *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*: 46–57.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. (2021). "Learning Transferable Visual Models From Natural Language Supervision". URL: https://arxiv.org/abs/2103.00020.

Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang. (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. DOI: 10.18653/v1/D16-1264.

Ramesh, A., M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. (2021). "Zero-shot text-to-image generation". In: *ICML*.

Rombach, R., A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. (2022). "High-Resolution Image Synthesis With Latent Diffusion Models". In: *CVPR*.

Sadigh, D., E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. (2014). "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications". In: *53rd IEEE Conference on Decision and Control*. IEEE. 1091–1096.

Saeed, M., N. Ahmadi, P. Nakov, and P. Papotti. (2021). "RuleBERT: Teaching Soft Rules to Pre-Trained Language Models". In: *EMNLP*.

Santoro, A., D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. (2017). "A simple neural network module for relational reasoning". *NeurIPS*.

Schick, T., J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. (2023). "Toolformer: Language Models Can Teach Themselves to Use Tools". arXiv: 2302.04761 [cs.CL].

Shang, X., D. Di, J. Xiao, Y. Cao, X. Yang, and T.-S. Chua. (2019). "Annotating Objects and Relations in User-Generated Videos". In: *Proceedings of the 2019 on International Conference on Multimedia Retrieval*. ACM. 279–287.

Shang, X., T. Ren, J. Guo, H. Zhang, and T.-S. Chua. (2017). "Video visual relation detection". In: *Proceedings of the 25th ACM international conference on Multimedia*. 1300–1308.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. (2016). "Mastering the game of Go with deep neural networks and tree search". *Nature*. 529: 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Sinha, K., S. Sodhani, J. Dong, J. Pineau, and W. L. Hamilton. (2019). "CLUTRR: A Diagnostic Benchmark for Inductive Reasoning from Text". arXiv: 1908.06177.

Soares, L. B., N. Fitzgerald, J. Ling, and T. Kwiatkowski. (2019). "Matching the Blanks: Distributional Similarity for Relation Learning". In: *ACL*.

Srivastava, A., A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, and et al. (2023). "Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models". arXiv: 2206.04615.

Sun, J. J., M. Tjandrasuwita, A. Sehgal, A. Solar-Lezama, S. Chaudhuri, Y. Yue, and O. Costilla-Reyes. (2022). "Neurosymbolic Programming for Science". URL: https://arxiv.org/abs/2210.05050.

Tenney, I., D. Das, and E. Pavlick. (2019). "BERT Rediscovers the Classical NLP Pipeline". In: *ACL*.

Thomee, B., D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. (2016). "YFCC100M: The New Data in Multimedia Research". *Communications of the ACM*. 59(2): 64–73.

Touvron, H., L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.* (2023). "Llama 2: Open Foundation and Fine-Tuned Chat Models". arXiv: 2307.09288.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. (2018). "Graph Attention Networks". In: *ICLR*.

Wang, P.-W., P. L. Donti, B. Wilder, and Z. Kolter. (2019). "SATNet: Bridging Deep Learning and Logical Reasoning Using a Differentiable Satisfiability Solver". In: *International Conference on Machine Learning (ICML)*. arXiv: 1905.12149.

Watkins, C. J. C. H. (1989). "Learning from delayed rewards".

Wu, Y., M. Keoliya, K. Chen, N. Velingker, Z. Li, E. J. Getzen, Q. Long, M. Naik, R. B. Parikh, and E. Wong. (2024). "DISCRET: Synthesizing Faithful Explanations For Treatment Effect Estimation". URL: https://arxiv.org/abs/2406.00611.

Yang, J., W. Peng, X. Li, Z. Guo, L. Chen, B. Li, Z. Ma, K. Zhou, W. Zhang, C. C. Loy, and Z. Liu. (2023). "Panoptic Video Scene Graph Generation". arXiv: 2311.17058 [cs.CV].

Yang, Z., P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. (2018). "HotpotQA: A dataset for diverse, explainable multi-hop question answering". arXiv: 1809.09600.

Yi, K., J. Wu, C. Gan, A. Torralba, P. Kohli, and J. Tenenbaum. (2018). "Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding". In: *Conference on Neural Information Processing Systems (NeurIPS)*.

Zhai, X., B. Mustafa, A. Kolesnikov, and L. Beyer. (2023). "Sigmoid Loss for Language Image Pre-Training". arXiv: 2303.15343 [cs.CV].

Zhu, C., Z. Li, A. Xue, A. P. Bajaj, W. Gibbs, Y. Liu, R. Alur, T. Bao, H. Dai, A. Doupé, M. Naik, Y. Shoshitaishvili, R. Wang, and A. Machiry. (2024). "TYGR: Type Inference on Stripped Binaries using Graph Neural Networks". In: *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association. 4283–4300. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/zhu-chang.

Zhu, G., L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, M. Feng, X. Zhao, Q. Miao, S. A. A. Shah, *et al.* (2022). "Scene graph generation: A comprehensive survey". *arXiv preprint arXiv:2201.00443*.