maze-dataset: Maze Generation with Algorithmic Variety and Representational Flexibility

Michael Igorevich Ivanitskiy [1¶], **Aaron Sandoval** [4], **Alex F. Spies** [2], **Tilman Räuker** [3], **Brandon Knutson** [1], **Cecilia Diniz Behn** [1], and **Samy Wu Fung** [1]

1 Colorado School of Mines, Department of Applied Mathematics and Statistics **2** Imperial College London **3** UnSearch.org **4** Independent ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Solving mazes is a classic problem in computer science and artificial intelligence, and humans have been constructing mazes for thousands of years. Although finding the shortest path through a maze is a solved problem, this very fact makes it an excellent testbed for studying how machine learning algorithms solve problems and represent spatial information. We introduce `maze-dataset`, a user-friendly Python library for generating, processing, and visualizing datasets of mazes. This library supports a variety of maze generation algorithms providing mazes with or without loops, mazes that are connected or not, and many other variations. These generation algorithms can be configured with various parameters, and the resulting mazes can be filtered to satisfy desired properties. Also provided are tools for converting mazes to and from various formats suitable for a variety of neural network architectures, such as rasterized images, tokenized text sequences, and various visualizations. As well as providing a simple interface for generating, storing, and loading these datasets, `maze-dataset` is extensively tested, type hinted, benchmarked, and documented.

The Journal of Open Source Software

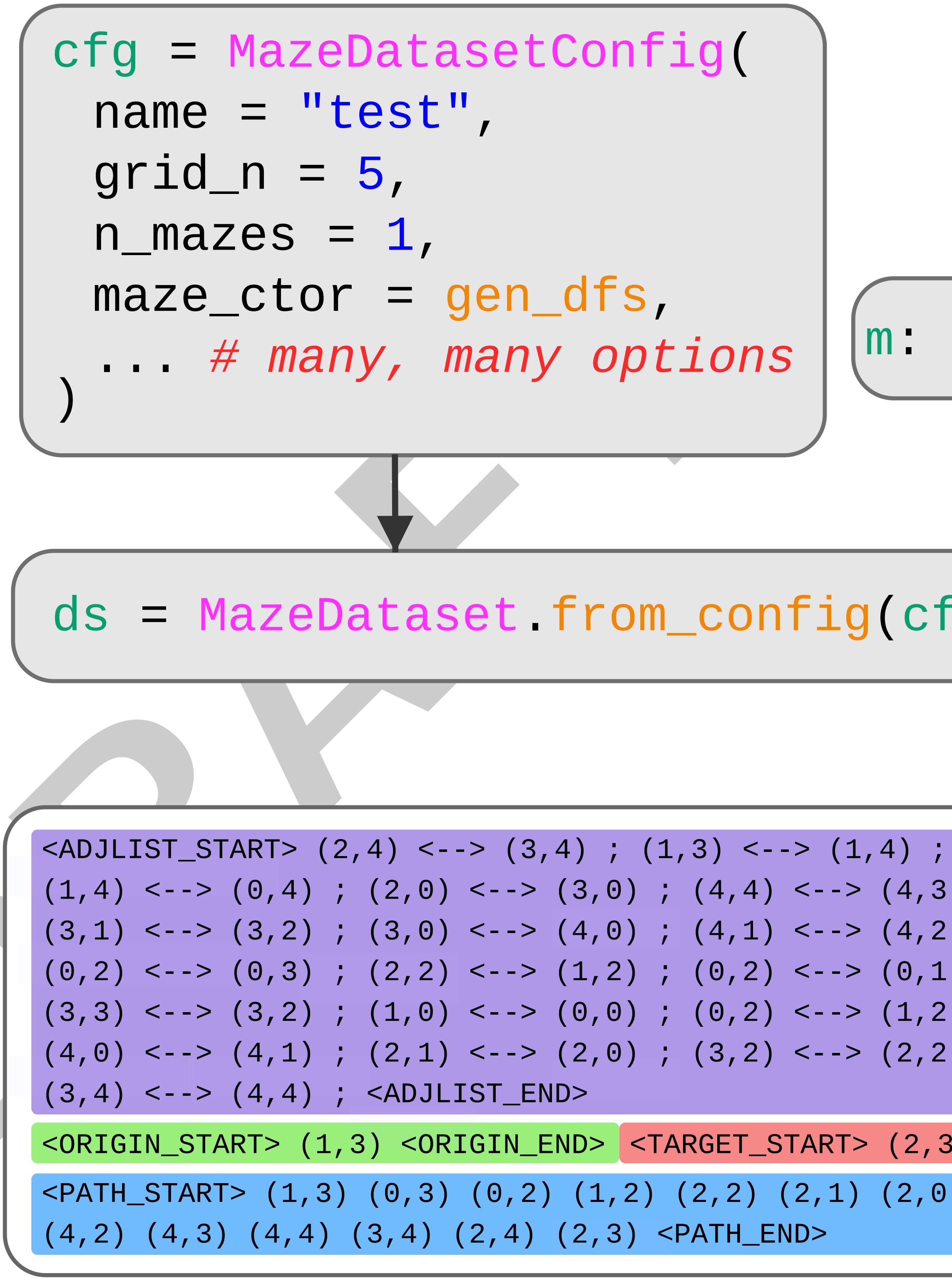


Figure 1: Usage of maze-dataset. We create a `MazeDataset` from a `MazeDatasetConfig`. This contains `SolvedMaze` objects which can be converted to and from a variety of formats. Code in the image contains clickable links to documentation. A variety of generated examples can be viewed here.

Statement of Need

While maze generation itself is straightforward, the architectural challenge comes from building a system supporting many algorithms with configurable parameters, property filtering, and representation transformation. This library aims to greatly streamline the process of generating and working with datasets of mazes that can be described as subgraphs of an $n \times n$ lattice with boolean connections and, optionally, start and end points that are nodes in the graph. Furthermore, we place emphasis on a wide variety of possible text output formats aimed at evaluating the spatial reasoning capabilities of Large Language Models (LLMs) and other text-based transformer models.

For interpretability and behavioral research, algorithmic tasks offer benefits by allowing systematic data generation and task decomposition, as well as simplifying the process of circuit discovery (Räuker et al., 2023). Although mazes are well suited for these investigations, we found that existing maze generation packages (Cobbe et al., 2019; Ehsan, 2022; Harries et al., n.d.; Németh, 2019; Schwarzschild, Borgnia, Gupta, Bansal, et al., 2021) lack support for transforming between multiple representations and provide limited control over the maze generation process.

Related Works

A multitude of public and open-source software packages exist for generating mazes (Ehsan, 2022; Németh, 2019; Schwarzschild, Borgnia, Gupta, Bansal, et al., 2021). However, nearly all of these packages produce mazes represented as rasterized images or other visual formats rather than the underlying graph structure, and this makes it difficult to work with these datasets.

- Most prior works provide mazes in visual or raster formats, and we provide a variety of similar output formats:
  - `RasterizedMazeDataset`, utilizing `as_pixels()`, which can exactly mimic the outputs provided in `easy-to-hard-data` (Schwarzschild, Borgnia, Gupta, Bansal, et al., 2021) and can be configured to be similar to the outputs of Németh (2019)
  - `as_ascii()` provides a format similar to (Oppenheim, 2018; Singla, 2023)
  - `MazePlot` provides a feature-rich plotting utility with support for multiple paths, heatmaps over positions, and more. This is similar to the outputs of (Alance AB, 2019; Ehsan, 2022; Guo et al., 2011; Nag, 2020)
- The text format provided by `SolvedMaze(...).as_tokens()` is similar to that of (Liu & Wu, 2023) but with many more options, detailed in section: *Tokenized Output Formats*.
- Preserving metadata about the generation algorithm with the dataset itself is essential for studying the effects of distributional shifts. Our package efficiently stores the dataset along with its metadata in a single human-readable file (M. Ivanitskiy, n.d.). As far as we are aware, no existing packages do this reliably.
- Storing mazes as images or adjacency matrices is not only difficult to work with, but also inefficient. We use a highly efficient method detailed in section: *Implementation*.
- Our package is easily installable with source code freely available. It is extensively tested, type hinted, benchmarked, and documented. Many other maze generation packages lack this level of rigor and scope, and some (Ayaz et al., 2008) appear to simply no longer be accessible.

Features

We direct readers to our examples, docs, and notebooks for more information. Our package can be installed from PyPi via `pip install maze-dataset`, or directly from the git repository (Michael I. Ivanitskiy et al., 2023a).

Datasets of mazes are created from a `MazeDatasetConfig` configuration object, which allows specifying the number of mazes, their size, the generation algorithm, and various parameters for the generation algorithm. Datasets can also be filtered after generation to satisfy certain properties. Custom filters can be specified, and some filters are included in `MazeDatasetFilters`.

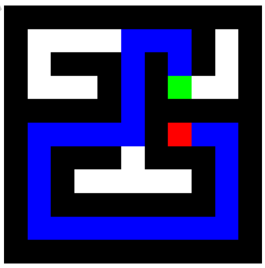
Visual Output Formats

Internally, mazes are `SolvedMaze` objects, which have path information and a tensor optimized for storing sub-graphs of a lattice. These objects can be converted to and from several formats, shown in Figure 2, to maximize their utility in different contexts.

In previous work, maze tasks have been used with Recurrent Convolutional Neural Network (RCNN) derived architectures (Schwarzschild, Borgnia, Gupta, Huang, et al., 2021). To facilitate the use of our package in this context, we replicate the format of (Schwarzschild, Borgnia, Gupta, Bansal, et al., 2021) and provide the `RasterizedMazeDataset` class which returns rasterized pairs of (input, target) mazes as shown in Figure 3.

`as_ascii()`	`as_pixels()`	`MazePlot()`
Simple text format for displaying mazes, useful for debugging in a terminal environment.	`numpy` array of `dtype=uint8` and shape `(height, width, 3)`. The last dimension is RGB color.	feature-rich plotting utility with support for multiple paths, heatmaps over positions, and more.

```
# # # # # # # # # # #
#         X X X #   #
#   # # # X # X #   #
#       # X # S     #
# # # # # X # # # # #
# X X X X X # E X X #
# X # # #   # # # X #
# X #           # X #
# X # # # # # # # X #
# X X X X X X X X X #
# # # # # # # # # # #
```



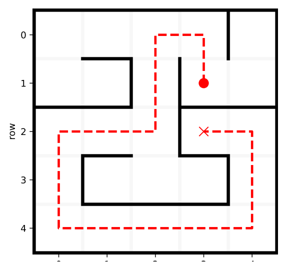


Figure 2: Various output formats. Top row (left to right): ASCII diagram, rasterized pixel grid, and advanced display tool.

The Journal of Open Source Software

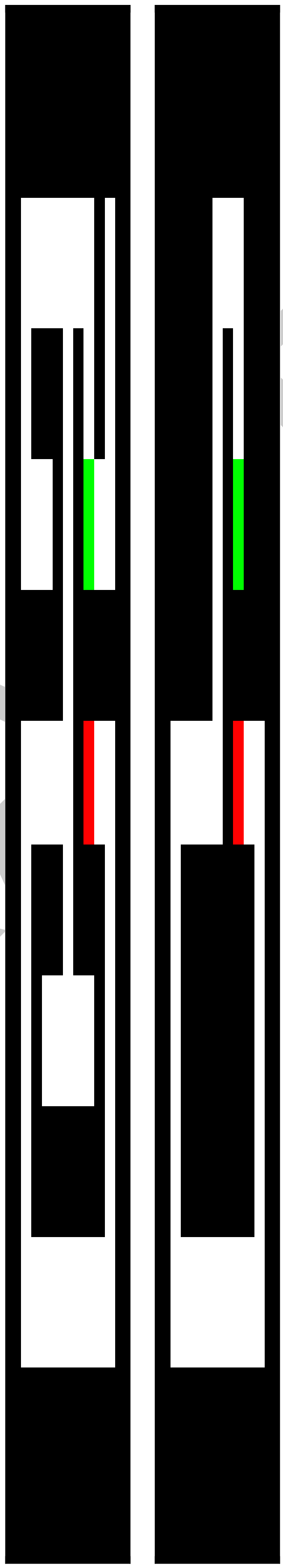


Figure 3: Input is the rasterized maze without the path marked (left), and provide as a target the maze with all but the correct path removed (right). Configuration options exist to adjust whether endpoints are included and if empty cells should be filled in.

Tokenized Output Formats

Autoregressive transformer models can be quite sensitive to the exact format of input data, and may even use delimiter tokens to perform reasoning steps (Pfau et al., 2024; Spies et al., 2024). To facilitate systematic investigation of the effects of different representations of data on text model performance, we provide a variety of text output formats, with an example given in Figure 4. We utilize Finite State Transducers (Gallant, 2015) for efficiently storing valid tokenizers.

```
<ADJLIST_START> (0,0) <--> (1,0) ; (2,0) <--> (3,0) ; (4,1) <--> (4,0) ; (2,0) <--> (2,1) ;
(1,0) <--> (1,1) ; (3,4) <--> (2,4) ; (4,2) <--> (4,3) ; (0,0) <--> (0,1) ; (0,3) <--> (0,2) ;
(4,4) <--> (3,4) ; (4,3) <--> (4,4) ; (4,1) <--> (4,2) ; (2,1) <--> (2,2) ; (1,4) <--> (0,4) ;
(1,2) <--> (0,2) ; (2,4) <--> (2,3) ; (4,0) <--> (3,0) ; (2,2) <--> (3,2) ; (1,2) <--> (2,2) ;
(1,3) <--> (0,3) ; (3,2) <--> (3,3) ; (0,2) <--> (0,1) ; (3,1) <--> (3,2) ; (1,3) <--> (1,4) ;
<ADJLIST_END>   <ORIGIN_START> (1,3) <ORIGIN_END>   <TARGET_START> (2,3) <TARGET_END>
<PATH_START> (1,3) (0,3) (0,2) (1,2) (2,2) (2,1) (2,0) (3,0) (4,0) (4,1) (4,2) (4,3) (4,4)
(3,4) (2,4) (2,3) <PATH_END>
```

Figure 4: Example text output format with token regions highlighted. Adjacency list : text representation of the graph, Origin : starting coordinate, Target : ending coordinate, Path : maze solution sequence. By passing an instance of `MazeTokenizerModular` to `as_tokens(...)` a maze can be converted to a text sequence. The `MazeTokenizerModular` class contains a rich set of options with 19 discrete parameters, resulting in over 5.8 million unique possible tokenizers.

Benchmarks

We benchmarks for generation time across various configurations in **??** and Figure 5. Experiments were performed on a standard GitHub runner without parallelism. Additionally, maze generation under certain constraints may not always be successful, and for this we provide a way to estimate the success rate of a given configuration, described in **??**.

The Journal of Open Source Software

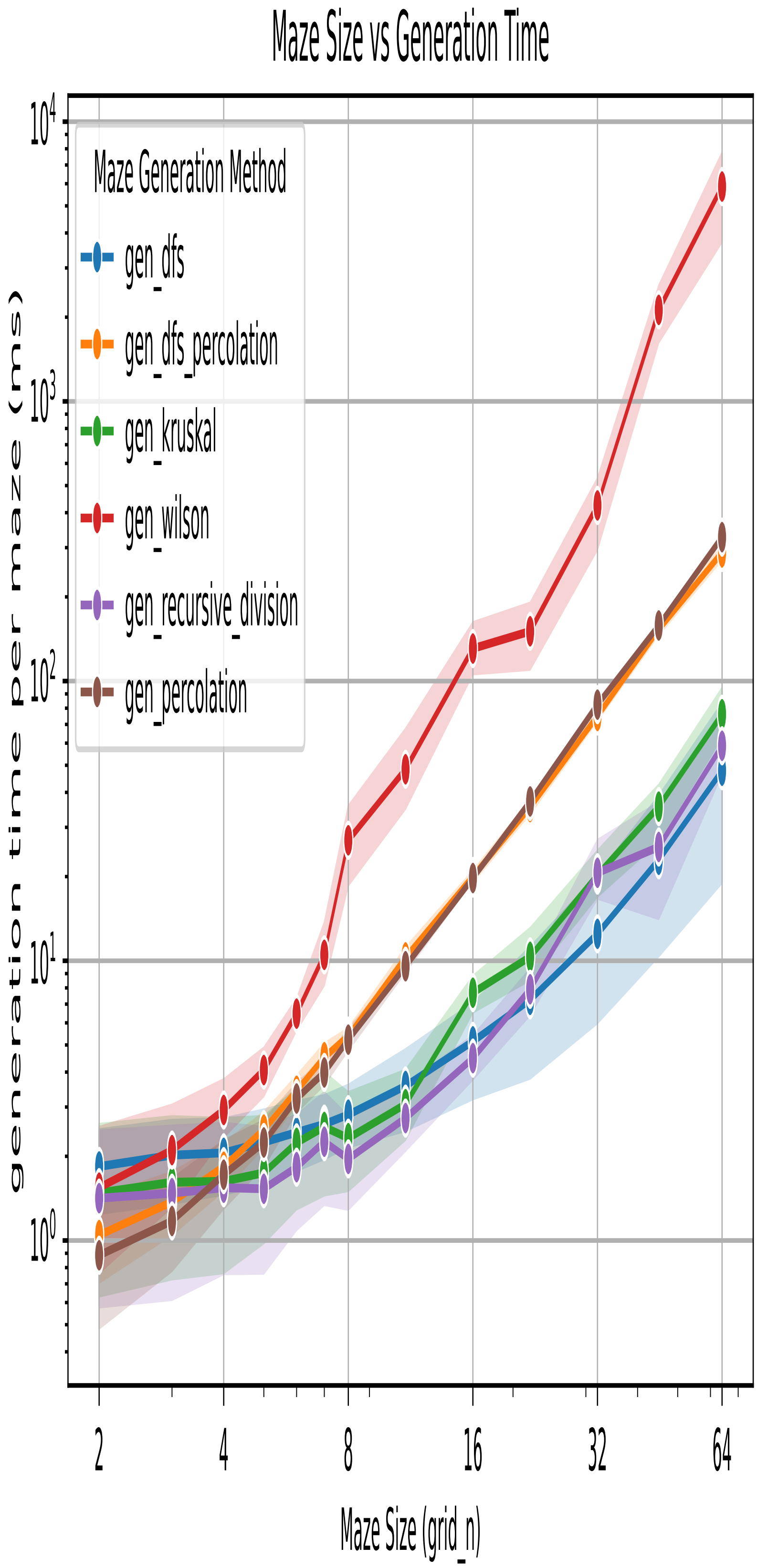


Figure 5: Plot of maze generation time. Generation time scales exponentially with maze size for all algorithms. Generation time per maze does not depend on the number of mazes being generated, and there is minimal overhead to initializing the generation process for a small dataset. Wilson's algorithm is notably less efficient than others and has high variance. Note that values are averaged across all parameter sets for that algorithm. More information can be found on the benchmarks page.

Implementation

Using an adjacency matrix for storing mazes would be memory inefficient by failing to exploit the highly sparse structure, while using an adjacency list could lead to a poor lookup time. This package utilizes a simple, efficient representation of mazes as subgraphs of a finite lattice, detailed in **??**, which we call a `LatticeMaze`.

Our package is implemented in Python(Rossum, 1995), and makes use of the extensive scientific computing ecosystem, including NumPy (Harris et al., 2020) for array manipulation, plotting tools (Hunter, 2007; Waskom, 2021), Jupyter notebooks (Kluyver et al., 2016), and PySR (Cranmer, 2023) for symbolic regression.

Usage in Research

This package was originally built for the needs of the (Michael I. Ivanitskiy et al., 2023b) project, which aims to investigate spatial planning and world models in autoregressive transformer models trained on mazes (Michael Igorevich Ivanitskiy, Spies, et al., 2023; Michael Igorevich Ivanitskiy, Shah, et al., 2023; Spies et al., 2024). It was extended for work on understanding the mechanisms by which recurrent convolutional and implicit networks (Fung et al., 2022) solve mazes given a rasterized view (Knutson et al., 2024), which required matching the pixel-padded and endpoint constrained output format of (Schwarzschild, Borgnia, Gupta, Bansal, et al., 2021). Ongoing work using `maze-dataset` aims to investigate the effects of varying the tokenization format on the performance of pretrained LLMs on spatial reasoning.

This package has also been utilized in work by other groups:

- By (Nolte et al., 2024) to compare the effectiveness of transformers trained with the MLM-$\mathcal{U}$ (Kitouni et al., 2024) multistep prediction objective against standard autoregressive training for multi-step planning on our maze task.
- By (Wang et al., 2024) and (Chen et al., 2024) to study the effectiveness of imperative learning.
- By (Zhang et al., 2025) to introduce a novel framework for reasoning diffusion models.
- By (Dao & Vu, 2025) to improve spatial reasoning in LLMs with GRPO.

Acknowledgements

This work was partially funded by National Science Foundation awards DMS-2110745 and DMS-2309810. We are also grateful to LTFF and FAR Labs for hosting authors MII, AFS, and TR for a residency visit, and to various members of FAR's technical staff for their advice.

This work was partially supported by AI Safety Camp and AI Safety Support, which also brought many of the authors together. We would like to thank our former collaborators at AI Safety Camp and other users and contributors to the `maze-dataset` package: Benji Berczi, Guillaume Corlouer, William Edwards, Leon Eshuijs, Chris Mathwin, Lucia Quirke, Can Rager, Adrians Skapars, Rusheb Shah, Johannes Treutlein, and Dan Valentine.

We thank the Mines Optimization and Deep Learning group (MODL) for fruitful discussions. We also thank Michael Rosenberg for recommending the usage of Finite State Transducers for storing tokenizer validation information.

References

Alance AB. (2019). *Maze generator*. http://www.mazegenerator.net.

Ayaz, H., Allen, S. L., Platek, S. M., & Onaral, B. (2008). Maze suite 1.0: A complete set of tools to prepare, present, and analyze navigational and spatial cognitive neuroscience experiments. *Behavior Research Methods*, *40*, 353–359. https://doi.org/10.3758/brm.40.1.353

Chen, X., Yang, F., & Wang, C. (2024). iA*: Imperative learning-based A* search for pathfinding. *arXiv Preprint arXiv:2403.15870*. https://doi.org/10.48550/arXiv.2403.15870

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2019). Leveraging procedural generation to benchmark reinforcement learning. *arXiv Preprint arXiv:1912.01588*. https://doi.org/10.48550/arXiv.1912.01588

Cranmer, M. (2023). Interpretable machine learning for science with PySR and SymbolicRegression. jl. *arXiv Preprint arXiv:2305.01582*. https://doi.org/10.48550/arXiv.2305.01582

Dao, A., & Vu, D. B. (2025). AlphaMaze: Enhancing large language models' spatial intelligence via GRPO. *arXiv Preprint arXiv:2502.14669*. https://doi.org/10.48550/arXiv.2502.14669

Ehsan, E. (2022). *Maze*. https://github.com/emadehsan/maze

Fung, S. W., Heaton, H., Li, Q., McKenzie, D., Osher, S., & Yin, W. (2022). Jfb: Jacobian-free backpropagation for implicit networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*, 6648–6656. https://doi.org/10.1609/aaai.v36i6.20619

Gallant, A. (2015). *Index 1,600,000,000 keys with automata and rust*. https://burntsushi.net/transducers/.

Guo, C., Barthelet, L., & Morris, R. (2011). *Maze generator and solver*. Wolfram Demonstrations Project, https://demonstrations.wolfram.com/MazeGeneratorAndSolver/.

Harries, L., Lee, S., Rzepecki, J., Hofmann, K., & Devlin, S. (n.d.). MazeExplorer: A Customisable 3D Benchmark for Assessing Generalisation in Reinforcement Learning. *2019 IEEE Conf. Games CoG*, 1–4. https://doi.org/10.1109/cig.2019.8848048

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., & al., et. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science and Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Ivanitskiy, M. (n.d.). *ZANJ*. https://doi.org/10.5281/zenodo.15540393

Ivanitskiy, Michael I., Shah, R., Spies, A. F., Räuker, T., Valentine, D., Rager, C., Quirke, L., Corlouer, G., & Mathwin, C. (2023a). *Maze dataset*. https://doi.org/10.48550/arXiv.2309.10498

Ivanitskiy, Michael I., Shah, R., Spies, A. F., Räuker, T., Valentine, D., Rager, C., Quirke, L., Corlouer, G., & Mathwin, C. (2023b). *Maze transformer interpretability*. https://doi.org/10.48550/arXiv.2312.02566

Ivanitskiy, Michael Igorevich, Shah, R., Spies, A. F., Räuker, T., Valentine, D., Rager, C., Quirke, L., Mathwin, C., Corlouer, G., Behn, C. D., & others. (2023). A configurable library for generating and manipulating maze datasets. *arXiv Preprint arXiv:2309.10498*. https://doi.org/10.48550/arXiv.2309.10498

Ivanitskiy, Michael Igorevich, Spies, A. F., Räuker, T., Corlouer, G., Mathwin, C., Quirke, L., Rager, C., Shah, R., Valentine, D., Behn, C. D., & others. (2023). Structured world representations in maze-solving transformers. *arXiv Preprint arXiv:2312.02566*.

https://doi.org/10.48550/arXiv.2312.02566

Kitouni, O., Nolte, N. S., Williams, A., Rabbat, M., Bouchacourt, D., & Ibrahim, M. (2024). The factorization curse: Which tokens you predict underlie the reversal curse and more. *Advances in Neural Information Processing Systems*, *37*, 112329–112355. https://doi.org/10.48550/arXiv.2406.05183

Kluyver, T., Ragan-Kelley, B., Perez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). Jupyter notebooks - a publishing format for reproducible computational workflows. *Proceedings of the 20th International Conference on Electronic Publishing*, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

Knutson, B., Rabeendran, A. C., Ivanitskiy, M., Pettyjohn, J., Diniz-Behn, C., Fung, S. W., & McKenzie, D. (2024). On logical extrapolation for mazes with recurrent and implicit networks. *arXiv Preprint arXiv:2410.03020*. https://doi.org/10.48550/arXiv.2410.03020

Liu, C., & Wu, B. (2023). Evaluating large language models on graphs: Performance insights and comparative analysis. *arXiv Preprint arXiv:2308.11224*. https://doi.org/10.48550/arXiv.2308.11224

Nag, A. (2020). MDL suite: A language, generator and compiler for describing mazes. *Journal of Open Source Software*, *5*(46), 1815. https://doi.org/10.21105/joss.01815

Németh, F. (2019). *Maze-generation-algorithms*. https://github.com/ferenc-nemeth/maze-generation-algorithms

Nolte, N., Kitouni, O., Williams, A., Rabbat, M., & Ibrahim, M. (2024). Transformers can navigate mazes with multi-step prediction. *arXiv Preprint arXiv:2412.05117*. https://doi.org/10.48550/arXiv.2412.05117

Oppenheim, J. (2018). *Maze-generator: Generate a random maze represented as a 2D array using depth-first search*. https://github.com/oppenheimj/maze-generator/; GitHub.

Pfau, J., Merrill, W., & Bowman, S. R. (2024). Let's think dot by dot: Hidden computation in transformer language models. *arXiv Preprint arXiv:2404.15758*. https://doi.org/10.48550/arXiv.2404.15758

Räuker, T., Ho, A., Casper, S., & Hadfield-Menell, D. (2023). Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 464–483. https://doi.org/10.1109/satml54575.2023.00039

Rossum, G. van. (1995). *Python reference manual* (CS-R9525). Centrum voor Wiskunde; Informatica (CWI). https://ir.cwi.nl/pub/5008/05008D.pdf

Schwarzschild, A., Borgnia, E., Gupta, A., Bansal, A., Emam, Z., Huang, F., Goldblum, M., & Goldstein, T. (2021). *Datasets for Studying Generalization from Easy to Hard Examples* (No. arXiv:2108.06011). arXiv. https://doi.org/10.48550/arXiv.2108.06011

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., & Goldstein, T. (2021). Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, *34*, 6695–6706. https://doi.org/10.48550/arXiv.2106.04537

Singla, A. (2023). Evaluating ChatGPT and GPT-4 for visual programming. *arXiv Preprint arXiv:2308.02522*. https://doi.org/10.48550/arXiv.2308.02522

Spies, A. F., Edwards, W., Ivanitskiy, M. I., Skapars, A., Räuker, T., Inoue, K., Russo, A., & Shanahan, M. (2024). Transformers use causal world models in maze-solving tasks. *arXiv Preprint arXiv:2412.11867*. https://doi.org/10.48550/arXiv.2412.11867

Wang, C., Ji, K., Geng, J., Ren, Z., Fu, T., Yang, F., Guo, Y., He, H., Chen, X., Zhan, Z., &

others. (2024). Imperative learning: A self-supervised neural-symbolic learning framework for robot autonomy. *arXiv Preprint arXiv:2406.16087*. https://doi.org/10.48550/arXiv.2406.16087

Waskom, M. L. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Zhang, T., Pan, J.-S., Feng, R., & Wu, T. (2025). T-SCEND: Test-time scalable MCTS-enhanced diffusion model. *arXiv Preprint arXiv:2502.01989*. https://doi.org/10.48550/arXiv.2502.01989