




# NADI – Network Analysis and Data Integration with a Domain Specific Language

Gaurav Atreya <sup>1¶</sup>, Todd Steissberg <sup>2</sup>, and Patrick A. Ray <sup>1</sup>

<sup>1</sup> Department of Chemical and Environmental Engineering, University of Cincinnati, OH, USA <sup>2</sup> U. S. Army Engineer Research and Development Center (ERDC), Davis, CA, USA ¶ Corresponding author

DOI: [10.21105/joss.08655](https://doi.org/10.21105/joss.08655)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Mengqi Zhao](#) 

## Reviewers:

- [@jhwahlgemuth](#)
- [@lolpro11](#)
- [@amcandio](#)

Submitted: 18 June 2025

Published: 27 August 2025

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

We present the Network Analysis and Data Integration (NADI) System, a developing software framework designed to facilitate river data analysis. NADI System includes a Domain Specific Language (DSL) that has a succinct and readable syntax for network metadata analysis as well as a plugin system to run user-defined functions on each node or the whole network. Plugins provide seamless integration with other softwares and programming languages.

NADI System can be used from the Command Line Interface (CLI), Graphical User Interface (GUI) using the NADI Integrated Development Environment (IDE), as a Rust library, or as a Python library, allowing users to write their plugins or programs.

## Statement of need

Hydrological analysis sometimes consists of data that is related to points in the river, and frequently with relationships between upstream/downstream points (e.g., higher correlation, mass balance). Some analyses that benefit from the use of such relationships are: finding inconsistencies in the data, filling missing data, and visualization of metadata. There is a need for intelligent computational assistance on a network-based system to reduce the workload that further improves the efficiency and reproducibility of research in this field ([Rosenberg et al., 2020](#)). Specific hydrology-focused softwares ([Gironás et al., 2010](#); [Rossman & Van Zyl, 2010](#)) lack general applicability, while general-purpose programming languages might not have a succinct syntax. This highlights the need for a balanced approach that combines specificity to hydrological research questions with general capabilities.

Domain Specific Languages (DSLs) have several advantages including improved code readability and maintainability due to the tailored syntax and semantics ([Albuquerque et al., 2015](#); [Mernik et al., 2005](#)). Among DSLs that have been developed for networks or hydrology, Graphviz focuses on graph visualization ([Ellson et al., 2004](#); [Gansner & North, 2000](#)), but lacks analytical capabilities. Hydrolang offers both analysis and visualization tools tailored to hydrological applications, although it is for web-based platforms ([Erazo Ramirez et al., 2022](#)). Languages designed for grid-based spatial analysis ([Kuhn & Ballatore, 2015](#); [Pullar, 2001](#)) are ill-suited for handling values like streamflow, which exhibit partial spatial continuity along river courses, but do not fit neatly into traditional 2D/3D spatial frameworks.

We present the NADI System that can load a river network as a Rooted Tree Graph ([Deo, 2016](#)) — which is known to be one of the best ways to represent the river network ([Abed-Elmdoust et al., 2017](#); [Kuhn & Ballatore, 2015](#); [Rinaldo et al., 2006](#)) — and provide the DSL for network metadata analysis. Most components of the NADI System are written in Rust ([Klabnik & Nichols, 2023](#)) due to the memory safety ([Bugden & Alahmar, 2022](#); [Fulton & Chan, 2021](#); [Xu et al., 2021](#)), runtime performances ([Zhang et al., 2023](#)), and the macro system that gives

us the metaprogramming features necessary for the plugin development.

The figure below shows the GUI of NADI IDE, with the editor (left top), function help (left bottom), terminal (top right), network viewer, and attribute browser (right bottom). These panes can be managed in a tiling window style.

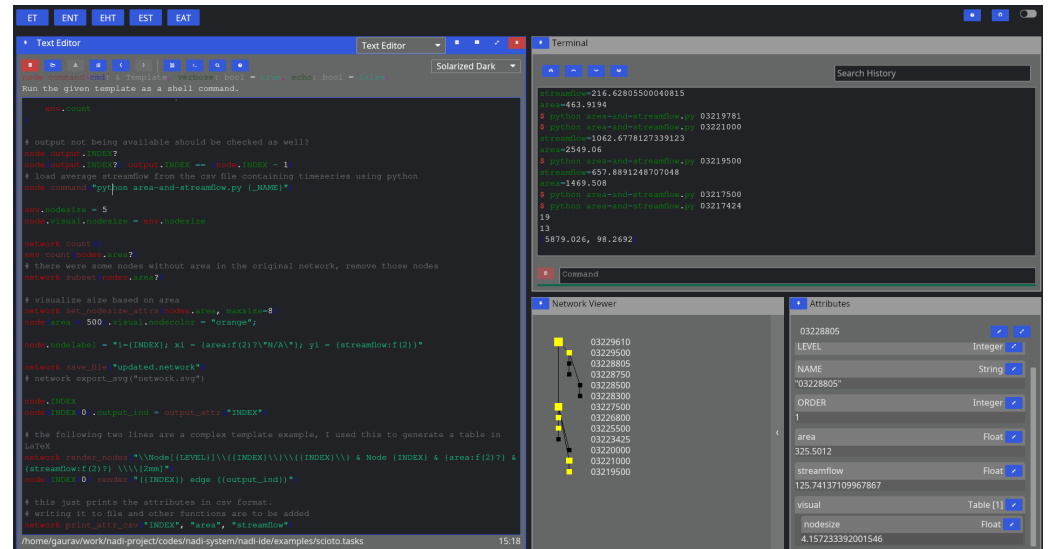


Figure 1: Screenshot of the NADI IDE showcasing different components

## Data Structures

The DSL is inspired by Python (Rossum & Drake, 2010), array programming, and Rust. Important components in the DSL are:

- **Node** is one point on the network. It can have input nodes, one output node, and attributes associated with it.
- **Network** consists of several connected nodes. It can also have attributes associated with it.
- **Attributes** are values that can be boolean, integer, float, string, date, time, datetime, array, and table.
- **Functions** are categorized into environment, network, and node functions based on what they work on. For example, network function is run on a network, while node function is run on each node.
- **Expressions** are a combination of attributes, variables, function calls, conditionals, etc, that can result in an attribute value.
- **Propagation**: As a node function is called on each node, Propagation determines which nodes are called and in which order.
- **Task** in NADI System is an execution body consisting of the type of task, optional output attribute name, and expression or function call. Only the top-level function call on the expression can be mutable (changes task context).
- **Task Context** is the runtime environment for the DSL. It stores network, all the variables, and functions from plugins.

The figure below shows how the DSL is run through different NADI applications and the internal structures of the task context. Each task runs in the task context, giving outputs, modifying the task context, or producing side effects (e.g., saving files).

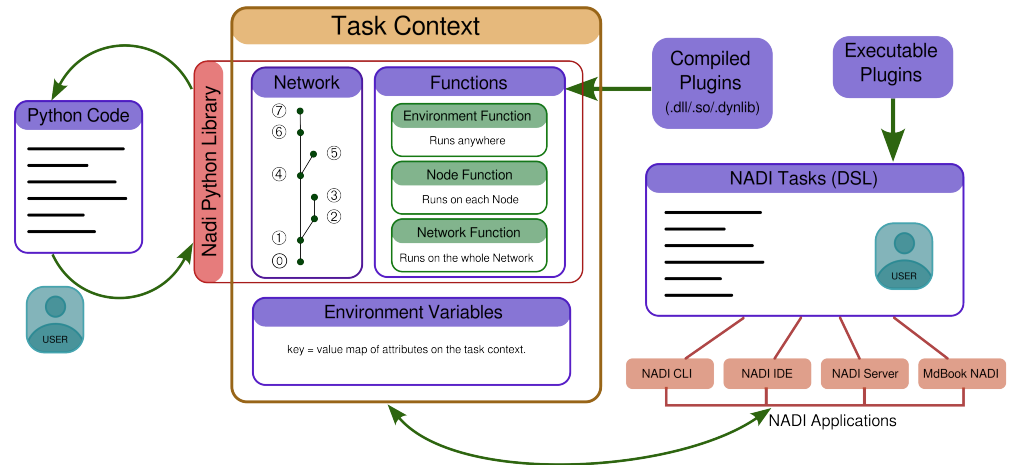


Figure 2: NADI Components and Internal Structure related to the DSL

## Network Analysis

Network Analysis is done through the Task System by loading Network and Attributes into the Task Context then running Tasks. For example, the following code represents a task that calculates the variable  $y$  as a cumulative sum of all the values of variable  $x$  at a node and its upstream points.

```
node<inputsfirst>.y = node.x + sum(inputs.y);
```

It is equivalent to:

$$y_i = x_i + \sum_{j \in I_i} y_j$$

Where  $x_i, y_i$  are values of  $x, y$  on node  $i$  and  $I_i$  is the set of input nodes for node  $i$ .

Atreya et al. (2024) demonstrates a complex task like river routing model using the network structure. For more concepts and up-to-date syntax, refer to the NADI Book (Atreya, 2025).

## Extensibility

NADI Task System supports two types of plugins for extending the use cases.

- **Compiled Plugins** are shared libraries (.so files in Linux, .dll in Windows, and .dylib in MacOS) containing a list of functions that can be loaded into the main program during runtime.
- **Executable Plugins** are independent programs that are run and their standard output is used to communicate values back to the NADI System.

Since DSLs have tradeoffs such as steep learning curves that can hinder adoption (Albuquerque et al., 2015), a Python library `nadi-py` is available to use the NADI Task System functions from Python (without the DSL).

Instructions on how to use them are available on the plugin developer guide and the Python library sections of the NADI Book (Atreya, 2025).

## Acknowledgements

Grant: #W912HZ-24-2-0049 Investigators: Ray, Patrick 09-30-2024 – 09-29-2025 U.S. Army Corps of Engineers Advanced Software Tools for Network Analysis and Data Integration (NADI) 74263.03 Hold Level: Federal

## References

- Abed-Elmdoust, A., Singh, A., & Yang, Z.-L. (2017). Emergent spectral properties of river network topology: An optimal channel network approach. *Scientific Reports*, 7(1), 11486. <https://doi.org/10.1038/s41598-017-11579-1>
- Albuquerque, D., Cafeo, B., Garcia, A., Barbosa, S., Abrahão, S., & Ribeiro, A. (2015). Quantifying usability of domain-specific languages: An empirical study on software maintenance. *Journal of Systems and Software*, 101, 245–259. <https://doi.org/10.1016/j.jss.2014.11.051>
- Atreya, G. (2025). *Network Analysis and Data Integration (NADI) System: User Manual* (0.7.0 ed.). <https://nadi-system.github.io/>
- Atreya, G., Emery, E., Rogacki, N., Buck, M., Soltanian, R., McAvoy, D., & Ray, P. (2024). Estimating the influence of water control infrastructure on natural low flow in complex reservoir systems: A case study of the Ohio River. *Journal of Hydrology: Regional Studies*, 54, 101897. <https://doi.org/10.1016/j.ejrh.2024.101897>
- Bugden, W., & Alahmar, A. (2022). The Safety and Performance of Prominent Programming Languages. *International Journal of Software Engineering and Knowledge Engineering*, 32(05), 713–744. <https://doi.org/10.1142/S0218194022500231>
- Deo, N. (2016). *Graph Theory with Applications to Engineering and Computer Science*. Courier Dover Publications. ISBN: 978-0-486-80793-5
- Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., & Woodhull, G. (2004). Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools. In G. Farin, H.-C. Hege, D. Hoffman, C. R. Johnson, K. Polthier, M. Jünger, & P. Mutzel (Eds.), *Graph Drawing Software* (pp. 127–148). Springer Berlin Heidelberg. [https://doi.org/10.1007/978-3-642-18638-7\\_6](https://doi.org/10.1007/978-3-642-18638-7_6)
- Erazo Ramirez, C., Sermet, Y., Molkenthin, F., & Demir, I. (2022). HydroLang: An open-source web-based programming framework for hydrological sciences. *Environmental Modelling & Software*, 157, 105525. <https://doi.org/10.1016/j.envsoft.2022.105525>
- Fulton, K. R., & Chan, A. (2021). *Benefits and Drawbacks of Adopting a Secure Programming Language: Rust as a Case Study*.
- Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11), 1203–1233. [https://doi.org/10.1002/1097-024X\(200009\)30:11%3C1203::AID-SPE338%3E3.0.CO;2-N](https://doi.org/10.1002/1097-024X(200009)30:11%3C1203::AID-SPE338%3E3.0.CO;2-N)
- Gironás, J., Roesner, L., Rossman, L., & Davis, J. (2010). A new applications manual for the Storm Water Management Model (SWMM). *Environmental Modelling & Software*, 25, 813–814. <https://doi.org/10.1016/j.envsoft.2009.11.009>
- Klabnik, S., & Nichols, C. (2023). *The Rust programming language* (2nd edition). No Starch Press. ISBN: 978-1-7185-0310-6
- Kuhn, W., & Ballatore, A. (2015). Designing a Language for Spatial Computing. In F. Bacao, M. Y. Santos, & M. Painho (Eds.), *AGILE 2015: Geographic Information Science as an Enabler of Smarter Cities and Communities* (pp. 309–326). Springer International Publishing. [https://doi.org/10.1007/978-3-319-16787-9\\_18](https://doi.org/10.1007/978-3-319-16787-9_18)
- Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific

- languages. *ACM Comput. Surv.*, 37(4), 316–344. <https://doi.org/10.1145/1118890.1118892>
- Pullar, D. (2001). MapScript: A Map Algebra Programming Language Incorporating Neighborhood Analysis. *Geoinformatica*, 5(2), 145–163. <https://doi.org/10.1023/A:1011438215225>
- Rinaldo, A., Banavar, J. R., & Maritan, A. (2006). Trees, networks, and hydrology. *Water Resources Research*, 42(6). <https://doi.org/10.1029/2005WR004108>
- Rosenberg, D. E., Fillion, Y., Teasley, R., Sandoval-Solis, S., Hecht, J. S., van Zyl, J. E., McMahon, G. F., Horsburgh, J. S., Kasprzyk, J. R., & Tarboton, D. G. (2020). The Next Frontier: Making Research More Reproducible. *Journal of Water Resources Planning and Management*, 146(6), 01820002. [https://doi.org/10.1061/\(ASCE\)WR.1943-5452.0001215](https://doi.org/10.1061/(ASCE)WR.1943-5452.0001215)
- Rossman, L., & Van Zyl, J. (2010). The open sourcing of EPANET. In *Water Distribution Systems Analysis 2010 - Proceedings of the 12th International Conference, WDSA 2010*. [https://doi.org/10.1061/41203\(425\)4](https://doi.org/10.1061/41203(425)4)
- Rossum, G. van, & Drake, F. L. (2010). *The Python language reference* (Release 3.0.1 [Repr.]). Python Software Foundation. ISBN: 978-1-4414-1269-0
- Xu, H., Chen, Z., Sun, M., Zhou, Y., & Lyu, M. R. (2021). Memory-Safety Challenge Considered Solved? An In-Depth Study with All Rust CVEs. *ACM Trans. Softw. Eng. Methodol.*, 31(1), 3:1–3:25. <https://doi.org/10.1145/3466642>
- Zhang, Y., Zhang, Y., Portokalidis, G., & Xu, J. (2023). Towards Understanding the Runtime Performance of Rust. *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 1–6. <https://doi.org/10.1145/3551349.3559494>