Evaluating a Visual Query Tracer and Builder for Learning Declarative Logic Programming

Julián Méndez*
Interactive Media Lab Dresden,
TU Dresden

Lukas Gerlach†
Knowledge-Based Systems Group,
TU Dresden

Tobias Wieland‡
Interactive Media Lab Dresden,
TU Dresden

Alex Ivliev§
Knowledge-Based Systems Group,
TU Dresden

Markus Krötzsch¶
Knowledge-Based Systems Group,
TU Dresden

Raimund Dachselt‖
Interactive Media Lab Dresden,
TU Dresden

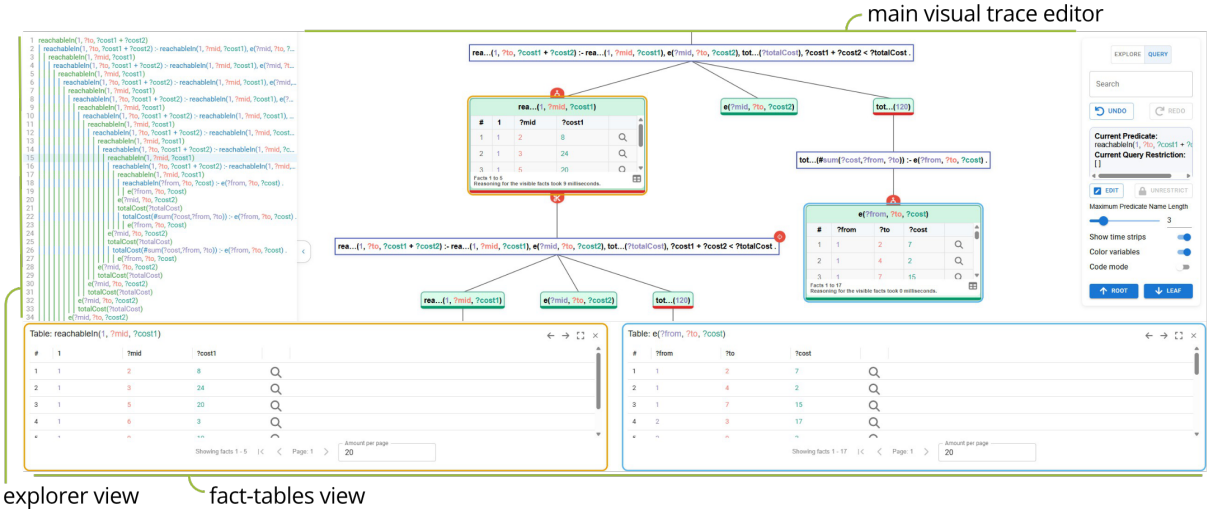

Figure 1: Screenshot of *Nemo Explain Visualizer* (*nev*), illustrating its three main interface components: the indented tree *explorer view* on the left, the *main visual trace editor* on the center, and *fact-tables view* at the bottom. The search bar, undo/redo, query restriction and other menu buttons are floating on the right.

Abstract

Nemo Explain Visualizer (nev) is an interactive visual query tracer and builder for Nemo, a powerful Datalog reasoner with extended features. Our tools were developed with and for expert users. However, considering the lack of resources to learn Datalog and similar declarative logic programming languages, we conducted a qualitative user study to assess how our tools might help students. The study, interviewing 14 participants with varying levels of involvement with the content of a university course on knowledge graphs, revealed a very positive assessment of our tools, which strengthens the value of visual explanation tools beyond their intended use.

Index Terms: Visual Traces, Visual Editing, Datalog, Declarative Logic Programming, User Study, Qualitative

1 Introduction

An important and diverse set of computing technologies is based on *rules* – simple *if-then* statements that express logical implications – which are combined into *logic programs*. The declarative rule language *Datalog* [28] is a core approach that recently received renewed interest for code analysis (e.g., Semmle/Github), database query (e.g., Google Logica), web data extraction, and more [25]. Being *declarative* means Datalog and its relatives let users focus on *what* should be achieved rather than on *how* it should be computed.

In the early 1990s, graph visualizations of logical computations were said to increase user understanding [38] and *derivation trees* to "closely mirror the manner in which users think of rule-based programs" [4]. This intuition still informs approaches to Datalog explanation and debugging [8, 36]. However, graph visualization is largely missing from modern logic programming IDEs [3, 5, 7, 9]. *Soufflé* [24] and *Nemo* [22] are modern Datalog reasoners that can compute derivation tree structures, but the only recent attempt to visualize them is a prototype we developed for Nemo [18]. Likewise, we lack user-centered evaluations of visual tools for Datalog reasoners. We must therefore ask whether visualizations in logic programming are as helpful and relevant as originally thought.

We tackled this question by (1) developing *Nemo Explain Visualizer* (*nev*), a web-based visual explanation tool for Datalog reasoning, and (2) conducting a qualitative user study to gather initial evidence of its benefits for declarative logic programming. Our Datalog environment consists of: (1) *Nemo*, the Datalog reasoner; (2) the *Nemo Web* frontend, with runtime controls and a rich program editor; and (3) *nev*, shown in Figure 1, which allows users to interactively manipulate query traces and retrieve computed facts on demand. *Nemo* and *Nemo Web* [22] received significant extensions to compute explorable explanation data on-demand, and *nev* is a new system developed to enable this exploration visually. All of our tools are free and open source (see https://imld.de/nev).

We designed *nev* in collaboration with domain *experts*, for de-

*e-mail: julian.mendez2@tu-dresden.de
†e-mail: lukas.gerlach@tu-dresden.de
‡e-mail: tobias.wieland@mailbox.tu-dresden.de
§e-mail: alex.ivliev@tu-dresden.de
¶e-mail: markus.kroetzsch@tu-dresden.de
‖e-mail: dachselt@acm.org

```
1  e(1,2,7).e(1,4,2).e(1,7,15).e(2,3,17).e(2,9,3).e(3,5,9).e(3,8,14).e(4,2,6).e(4,6,1).
2  e(5,1,8).e(5,10,2).e(6,3,13).e(6,9,3).e(7,5,5).e(8,2,4).e(9,7,2).e(9,10,9).
3
4  totalCost(#sum(?cost, ?from, ?to)) :- e(?from, ?to, ?cost).
5
6  reachableIn(?from, ?to, ?cost) :- e(?from, ?to, ?cost).
7  reachableIn(1, ?to, ?cost1+?cost2) :- reachableIn(1, ?mid, ?cost1), e(?mid, ?to, ?cost2), totalCost(?totalCost), ?cost1 + ?cost2 < ?totalCost.
8
9  minDistancesFromNode1(?to, #min(?cost)) :- reachableIn(1, ?to, ?cost), ?to != 1.
```

Figure 2: Screenshot of *Nemo Web* with a Datalog program that computes paths of minimal weight in a graph.

bugging and development. However, for our evaluation, we target mostly early but also advanced Datalog *learners*, to ensure a broader user group. The driving assumption behind this decision is that a well-designed system for programming also supports *computational thinking* [40], which is helpful for both learners and experts. Furthermore, involving users throughout the design process is important for human-centered design [17]. Therefore, despite the state of our tool (i.e., without specific learning features [2]), our evaluation with learners aims at making *nev* accessible to our broader target group, with the expert side already partially accounted for, because of our user-centered design process. We therefore conducted 14 interviews to study the value of our Datalog environment for explainability at lower expertise levels. The results of this study inform the prioritization of tool developments in nascent communities, since visual explanation tools might alleviate the need for tailored learning resources. More so, the results of our study support the intuition that visualization has significant potential in this domain, and therefore deserves renewed focus in the logic programming community.

2 Background & Related Work

This section provides basic notions of declarative logic programming, related visual tools, and theory on computational thinking.

2.1 Declarative Logic Programming

Declarative logic programming languages at their core consist of simple "if-then" rules. For example, the rule $B(x) \leftarrow A(x)$ means that if the fact $A(c)$ holds, then $B(c)$ is also true. In practice, it is common to use Datalog extensions to also support datatypes, negation, aggregation, or existentially quantified variables. Prominent examples of Datalog systems with subsets of these features include *RDFox* [31], *Soufflé* [24], *VLog* [10], and *Nemo* [22].

We can trace how any fact in the result of a Datalog program came into existence by backtracking the rules that contributed to the inference, all the way to the initial facts, resulting in a proof tree. This makes the systems inherently explainable. *Soufflé* and *Nemo* already include features to obtain such proof trees, also called traces, for a given fact in their result. Consider the example Datalog program from Figure 2, written in *Nemo Web*. It encodes a directed weighted graph with a ternary predicate `e(from, to, weight)` and stores the *transitive closure* of the edges in the `reachableIn` predicate. For example, the edges 1 to 2 with weight 7 and 2 to 9 with weight 3, imply that 9 is reachable from 1 with a combined weight of 10, stored as `reachableIn(1,9,10)` (line 7 in Figure 2). Tracing this derived fact, shows us its proof tree. In this case, we expect a tree that has `reachableIn(1,9,10)` at its root, with `edge(1,2,7)` and `edge(2,9,3)` as leaves.

2.2 Visual Tools to Explain, Trace, and Inspect Queries

Visualization has been widely employed in the field of knowledge representation and reasoning, e.g., for ontology analysis and editing [21, 6, 27] tasks. In areas closer related to logic programming, IDEs provide basic visualization features such as syntax highlighting [7, 9, 5] and tools like ASP Chef [3] even allow to build complex ASP programs by dragging and dropping basic building blocks. This also includes visualization capabilities for common structures, e.g., graphs, that are encoded in the program.

We refer to visualization tools concerned with explaining the outcomes of reasoning processes as *visual explanation tools*. Some examples include proof visualizers [1, 4, 29, 38] and database execution plan visualizers [11, 14, 19, 32]. The former focus on presentation and interactivity of the proofs, while the latter support debugging slow queries. The proofs/traces supported by these systems are often trees or directed acyclic graphs (DAGs), which may contain repeating patterns and recursive structures. Naturally, these tools employ techniques for visually exploring node-link diagrams [12], such as multiple coordinated views [34], and details on demand within nodes and onto juxtaposed views [23]. The survey by Dudáš et al. lists components typical to ontology visualization tools, such as radar views (overview), history (in case of editing), graphical zoom and pan, entity focus, and search and highlight [16]. Therefore, we used it to inform several features of *nev*.

2.3 Evaluating Visual Explanation Tools for Learning

We based the design of our evaluation on the concept of *computational thinking* (CT) [40], which refers to the employment of computing strategies such as decomposition and abstraction to tackle complex problems. Although CT has been discussed with various terminologies [37, 15] it is roughly described as a mixture of skills of data representation, algorithmic design, and pattern recognition. Since the subject of our study is declarative logic programming, it follows that a tool that supports CT in this computer science domain should support not only working on it, but also learning it. Furthermore, for human-centered design, users should be involved in early evaluations of the entire user experience (UX) [17]. We therefore evaluate CT and UX for our programming environment, as these assessments correspond to learners and experts simultaneously.

3 Nemo Explain Visualizer (nev)

The name and design of *nev* are heavily inspired by Postgres Explain Visualizer 2 (*pev2*) [14]. In this section, we explain our extension of this concept through the visual construction and manipulation of shape-based queries.

3.1 *Nemo* and the Tree-Shape Query

For a given fact, *Nemo* can compute a proof tree (trace) to witness how the fact came into existence during the reasoning process. We refer to these queries as type 1 (*T1*). The new type of query (*T2*) instead receives the *shape of a proof tree* and populates it with facts from all traces that include a partial proof tree of the given shape (if any exist). *T1* is useful for explaining individual facts, but *T2* was introduced mainly for debugging purposes to see how different rule applications interact without the need of changing the program. On the example from Figure 2, to know which facts of `reachableIn` correspond to paths with exactly three graph edges, one can build a *T2* with a tree shape of two applications of the rule in line 7 after one application of the rule in line 6. *Nemo* then returns the input shape with facts from trees matching this shape. In this case, the desired facts can be read on the root node. *T2* also includes execution times of applications to assist in analyzing performance. *T2*s are formally described for Nemo, and their performance measured, in [13].

3.2 Visual Analysis and Query Editing

We developed *nev* to simultaneously use the graphical trace for analysis and query editing, thanks to *T2*s using *Nemo* traces as input and output. Upon running a program on Nemo Web, the user is presented with tables that contain the derived facts for all rules. Choosing a derived fact opens *nev* with the respective *T1* in a new browser tab. The interface of *nev* consists of a *main visual trace editor*, an indented tree *explorer view* and *fact-tables view*, as shown in Figure 1. We distinguish two types of nodes: *predicates* using green highlights (i.e., border and fill color) and *rules* using blue borders. One can zoom and pan the main view for navigation, but hovering or clicking on the indented tree highlights and brings nodes to the center of the main view for faster navigation. Clicking on a predicate node in the main view reveals the facts associated with it on a small table embedded in the node. These tables can be pinned to the fact-tables panel at the bottom for inspection (using pagination and counts) as well as comparison e.g., for debugging purposes. Opposed to the modal dialog approach previously available to *Nemo Web* [18], *nev* is loaded onto a separate browser tab. The user can thus configure their screen(s) flexibly (e.g., side-by-side, different monitors) without cluttering the code editor view.

*T1*s establish a *query restriction* that specifies the facts to look for. Without it or upon manipulating the tree shape (i.e., adding or removing nodes), *T2*s are created. Users may *add* or *remove rules* using the green and red buttons above and below predicates nodes where this is viable (i.e., *T2*s include potential rules to add). When more than one rule can be added, the choices are shown via dialogs. Users can also *focus on a rule* by clicking the target icon on rule nodes. This constructs a *T2* of a single rule and its surrounding predicates, which is useful for inspecting all applications of a rule across all possible derivations. To support back and forth between the different traces corresponding to these features, we incorporated undo and redo features that save the tree (including restrictions) after any modification. To additionally mitigate the risk of accidental clicks, as well as to support incremental exploration, we also enable an *exploration mode* that disables editing, where users can collapse and expand the tree from any node, and focus on specific rules by blurring the rest of the tree instead of removing it. We also provide search and name shortening options.

4 Evaluation

We conducted a qualitative study of our tools on students with some level of involvement in declarative logic programming. As indicated in 2.3, this is motivated by the close relation between our use case and computational thinking (CT) [40] and the need to involve users in early design phases [17]. We conducted semi-structured interviews, following a think-aloud protocol. The research questions of this experiment [30] were: **RQ1:** How do the interactive editor features and visual tracing help students understand rule-based reasoning? **RQ2:** How does subject literacy (i.e., on declarative logic programming) influence the use of *Nemo* + *nev*? **RQ3:** How does the user experience and task difficulty scale with *Nemo* + *nev* with respect to task complexity? All research questions serve to validate *nev*, but RQ1 and RQ3 additionally address CT. The study was preregistered [30] and our supplementary material includes all used forms, documents, and anonymized data and results. The version of *Nemo Web* and *nev* used for this study is available online at tools.iccl.inf.tu-dresden.de/nemo/nev-study-2025.

Participants: We interviewed 14 participants (3 female, 11 male) enrolled in bachelor (3), master (4), or diploma (4) study programs. Regarding expertise, we used a scale from 0 (no experience) to 5 (expert). Only two participants scored themselves as Experts in *Nemo* and Advanced in Datalog. Every other score, including by the same participants in the other topics, was 3 (Intermediate) or below, with 4 participants averaging 1 or below (very low to no experience overall). Participation was compensated with 20 EUR.

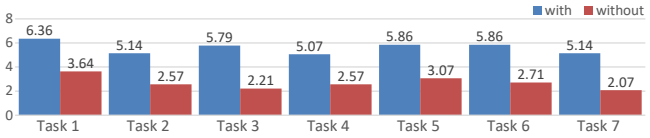

Figure 3: Average SEQ assessments for all tasks (1 = very difficult, 7 = very easy) for the *perceived* task difficulty using our tools (blue) and the *imagined* difficulty without our tools (red). Due to the absence of a baseline, these values support our tools only qualitatively.

Apparatus: The interviews were conducted in person on a laboratory setting with two interviewers per participant, one who guided the participants through the tasks, the other for taking notes but also intervening in case of questions. The participants would use one of two desktop computers with a 4K UHD 32" monitor, Intel i7 processors (-11700K and -13700K), and NVIDIA GeForce RTX 3080 and 4090, on the Brave browser on Windows 11.

Procedure: We collected informed consent, demographic data, audio, and screen captures. The experiment took in average 77 minutes, with a time limit of two hours that no participant exceeded. We devised a scenario where a faulty Datalog program must be iteratively corrected and improved until it corresponded to the program shown in Figure 2. This is an ecologically valid scenario common when developers leave behind pieces of code that no other employee maintained or looked at. The participants would work with this program over seven tasks, always in the same order, and could freely converse with the interviewers. The interviewers did not assist the participants in completing the tasks, but provided general hints about e.g., syntax and how Datalog rules work, upon request. Each task would be assessed in terms of task difficulty (TD) and user experience (UX). After the interviews were finished, we provided space for final general comments about our tools.

Tasks: The given program should compute the minimal cost between nodes in a weighted graph (10 nodes, 17 edges), but this information was withheld. The code initially contained syntactic errors, undefined variables, etc.. All participants would solve the following 7 tasks in order: 1) correct the breaking errors in the code; 2) inspect various traces to understand the program and rename symbols accordingly; 3) find paths of minimal cost and shorter paths (regardless of cost) using *nev* (without editing the program further); 4) inspect and correct a performance issue in the code; 5) modify the *nev* queries to find paths of specific length; and finally, assess the scalability of our tools by finding lowest cost paths on 6) larger (100 nodes, 444 edges), and 7) much larger graphs (500 nodes, 2284 edges). For the first task, only *Nemo Web* would be used, but for the other six, the participants could use both *nev* and *Nemo Web* freely. These tasks correspond to the CT principles and relation to user experience discussed in 2.3. Namely, identifying a graph data structure (tasks 1 and 2), modifying a recursive algorithm (1 and 4) and re-purposing features and interface components (3, 5). Tasks 6 and 7 directly address RQ3.

Data Collection: We used OBS Studio [26] for screen and audio capture. For subtitle generation/transcription we used Buzz [39] with Whisper [33]. Task difficulty (TD) was assessed using Single Ease Question (SEQ) [35] scales, while for user experience (UX) we used a modified version of the Short User Experience Questionnaire (UEQ-S) [20]. SEQs are 7-point Likert scales from *very difficult* to *very easy*, while UEQ-S is composed of eight 7-point Likert scales. For TD, we used two SEQs per task and asked the participants to rate the difficulty of solving the tasks using our tools (*Nemo Web* for task 1, both *Nemo Web* and *nev* for the other tasks) and the imagined difficulty of completing the tasks without our tools. For UX, we excluded the 7th question of the UEQ-S due to its similarity to the 8th. This helped us maintain a reasonable interview time

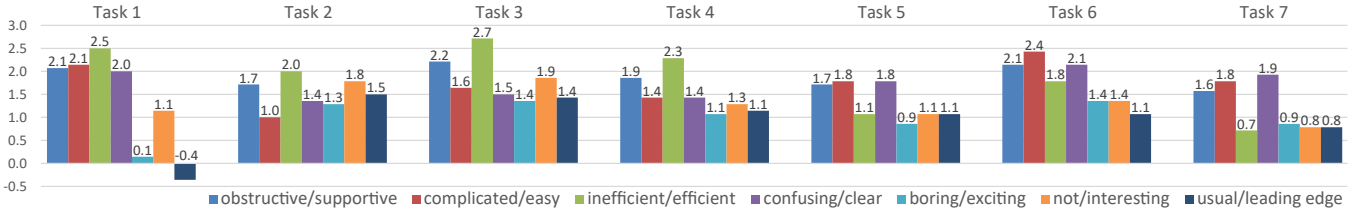

Figure 4: Average UX assessments for all tasks. UEQ-S items are normalized from -3 (horribly bad) to +3 (extremely good). Except for usual/leading edge for Task 1, all assessments are positive, in particular the pragmatic ones (the first 4 from left to right).

but precludes statistical conclusions about the hedonic nature of our tools. Nevertheless, the 7 questions remain meaningful descriptors of our tools. We took notes of the participant's comments, bugs encountered, and suggestions for features.

Results & Discussion: All participants completed all the tasks within the time limit. Figure 3 shows the average TD scores, while Figure 4 shows the equivalent for UX. Our tools were very positively received: e.g., "It was pretty intuitive, very nicely done", "It is helpful to have this tree with the derivations, and adding and removing rules, also restricting and unrestricting". Regarding TD, the SEQ scores show a very positive reception of our tools, which is unsurprising considering the lack of alternatives for declarative logic programming (e.g., text editors and command-line interfaces). When switching from just *Nemo Web* (task 1) to also using *nev* (tasks 2 and on), we received comments like "before using *nev* I would say this was moderately hard and after using *nev*, I think more easy". All averages (using our tools versus not using them) differ by at least 2.5 points, with the largest differences on tasks 3 (3.57), 6 (3.14), and 7 (3.07). These tasks correspond to the scalability comparison of RQ3, and a reasonable reduction of the difference can be noted as the graph size increases. We interpret these results as a positive indication of the scalability of our approach, despite ample room for further features to improve scalability. This interpretation is supported by the similarly high UX scores and the matching trend between tasks 3, 6, and 7. However, a mixed sentiment was expressed by some participants, e.g., "*nev* is probably not made to read a path along such a long derivation, but even then it works well" and "It doesn't scale very well. Still much better than just using *Nemo*" (referring to *nev*). Towards CT, these positive results are influenced by the interviewer assistance provided to alleviate the learning curve and limit the duration of the interviews, e.g., "Without any guidance it would be much more complicated to figure out, but I guess that's what tutorials are for". Even when acknowledging this, our participants showed positive sentiments, though: "once I understood what the buttons do, I can easily perform any task". Regarding UX, the UEQ-S questions were also overall very positive, with a single exception on Task 1. Since this task only introduced *Nemo Web* and its language server features that are common to many (more popular) programming languages, this is understandable. A participant indicated: "It follows what I expect an editor to look like (...) it's also boring, which I consider to be a good thing". In fact, the same task also received the second highest average score (for efficiency, highest on task 3), which was also emphasized by several participants, e.g., "I like the snappiness". With respect to RQ2, the participants with higher expertise had an easier time both completing the tasks and using the tools. This was reflected by their understanding of the program before completing Task 2. Namely, the most experienced participant identified the graph structure and purpose of the program during Task 1. Those with less experience would only understand the program (after hints by the interviewers) shortly before task 3. However, most participants were able to correctly answer validation questions asked throughout the interview, such as identifying equivalent ways to unrestrict the queries. Altogether, the assessments to TD and UX address RQ1, showing that our language server, visual tracing, query building, and performance inspection features are well-received for working with and learning declarative logic programs. Furthermore, the higher UX scores in pragmatic over hedonic qualities reflect the prioritization of feature developments from our side. During the interviews, several bugs were identified by the participants, such as on the queuing of requests when multiple pagination requests are triggered in quick succession, and with the replace symbol interaction on the code editor. These issues have been addressed in the respective repositories and several post-study updates are already reflected in the screenshot of Figure 1. The improvements include variable coloring, colored strips (green to red) to assess performance per node at a glance, and the ability to jump to and highlight lines in the code on the linked Nemo Web tab by clicking on the corresponding rules or focus buttons.

Limitations: The absence of a baseline for comparison (e.g., a command-line interface) precludes quantitative observations regarding the TD scores, which should therefore only be taken as evidence of subjective support for our tools. Furthermore, future evaluations of our tools should invite more participants that sample the entire target group (i.e., experts and students alike), gather feedback from educators, and quantitatively analyze learning benefits (e.g., measured comprehension, grades) in an unsupervised setting (i.e., without interviewer guidance). Furthermore, scalability has only been superficially discussed thus far. Theoretically, much larger traces than the ones in our study may be constructed. Thus, we aim to explore node grouping, focus+context techniques, and e.g., off-screen indicators of (infinite) recursion in future versions.

5 Conclusion

We presented *nev* and the results of an evaluation of our declarative logic programming environment. By designing our study in an incremental way around tasks that require understanding of algorithms and data structures, we also assessed the potential of our tools to support computational thinking. We received very positive assessments of our tools with respect to task difficulty and user experience. Our work illustrates the value that visual explanation tools can provide for communities with scarce learning resources instead of using general-purpose learning tools.

Acknowledgments

The authors thank Afshin Zanganeh for his contributions to nev. No AI was used in the creation of this paper or its contributions. This work is funded by Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy: EXC 2050/2, 390696704 – "Centre for Tactile Internet" (CeTI); by DFG grant 389792660 as part of TRR 248 – CPEC; by Bundesministerium für Bildung und Forschung (BMBF) and Saxon State Ministry for Science, Culture and Tourism (SMWK) in Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI, SCADS22B); and by BMBF and German Academic Exchange Service (DAAD) in project 57616814 (SECAI, School of Embedded and Composite AI).

References

[1] Y. Adachi and Y. Furusawa. Logichart: A prolog program diagram and its layout. *Electron. Commun. EASST*, 7, 9 2007. doi: 10.14279/TUJ.ECEASST.7.95 2

[2] J. S. Alqurni. Evaluating the user interface and usability approaches for e-learning systems. *Int. J. Inf. Technol. Web Eng.*, 18(1):1–25, 11 2023. doi: 10.4018/IJITWE.333638 2

[3] M. Alviano and L. A. Rodriguez Reiners. ASP Chef: Draw and Expand. In *Proc. Int. Conf. Principles Knowl. Represent. Reason.*, pp. 720–730, 8 2024. doi: 10.24963/kr.2024/68 1, 2

[4] T. Arora, R. Ramakrishnan, W. G. Roth, P. Seshadri, and D. Srivastava. Explaining program execution in deductive systems. In *Deductive and Object-Oriented Databases*, vol. 760 of *LNCS*, pp. 101–119. Springer, 1993. doi: 10.1007/3-540-57530-8_7 1, 2

[5] L. Bellomarini, A. Gentili, D. Magnanimi, and E. Sallinger. Vadacode: A logician-friendly IDE for Datalog+/-. *Proc. VLDB Endow.*, 18(12):5411–5414, 2025. doi: 10.14778/3750601.3750684 1, 2

[6] G. Braun, C. Gimenez, L. Cecchi, and P. Fillottrani. crowd: A visual tool for involving stakeholders into ontology engineering tasks. *KI - Künstl. Intell.*, 34:365–371, 5 2020. doi: 10.1007/s13218-020-00657-8 2

[7] P. Busoniu, J. Oetsch, J. Pührer, P. Skocovsky, and H. Tompits. SeaLion: An eclipse-based IDE for answer-set programming with advanced debugging support. *Theory Pract. Log. Program.*, 13(4-5):657–673, 2013. doi: 10.1017/S1471068413000410 1, 2

[8] R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. A theoretical framework for the declarative debugging of datalog programs. In *Semant. Data Knowl. Bases, 3rd Int. Workshop*, vol. 4925 of *LNCS*, pp. 143–159. Springer, 2008. doi: 10.1007/978-3-540-88594-8_8 1

[9] F. Calimeri, S. Germano, E. Palermiti, K. Reale, and F. Ricca. Developing ASP programs with ASPIDE and LoIDE. *KI - Künstl. Intell.*, 32(2-3):185–186, 2018. doi: 10.1007/S13218-018-0534-Z 1, 2

[10] D. Carral, I. Dragoste, L. González, C. Jacobs, M. Krötzsch, and J. Urbani. VLog: A rule engine for knowledge graphs. In *ISWC*, vol. 11779, pp. 19–35. Springer, 2019. doi: 10.1007/978-3-030-30796-7_2 2

[11] M. Christofides. pgMustard: Review Postgres query plans quickly. https://www.pgmustard.com/, 2026. Accessed: 06-2026. 2

[12] C. Collins and S. Carpendale. VisLink: Revealing Relationships Amongst Visualizations. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1192–1199, 11 2007. doi: 10.1109/TVCG.2007.70521 2

[13] R. Dachselt, L. Gerlach, P. Hanisch, A. Ivliev, M. Krötzsch, M. Marx, and J. Méndez. Declarative debugging for datalog with aggregation. In *Proc. Workshops EDBT/ICDT Joint Conf.*, CEUR Workshop Proceedings. CEUR-WS.org, 3 2026. https://ceur-ws.org/Vol-4192/TGD-paper4.pdf. 2

[14] Dalibo. Pev2. https://github.com/dalibo/pev2, 2026. Accessed: 06-2026. 2

[15] N. Dehbozorgi and M. Roopaei. Improving computational thinking competencies in stem higher education. In *IEEE Integr. STEM Educ. Conf.*, vol. 14, pp. 01–04, 2024. doi: 10.1109/ISEC61299.2024.10664997 2

[16] M. Dudáš, S. Lohmann, V. Svátek, and D. Pavlov. Ontology visualization methods and tools: a survey of the state of the art. *Knowl. Eng. Rev.*, 33:e10, 2018. doi: 10.1017/S0269888918000073 2

[17] M. Garreta-Domingo, D. Hernández-Leo, and P. B. Sloep. *Education, Technology and Design: A Much Needed Interdisciplinary Collaboration*, pp. 17–39. Springer, 2018. doi: 10.1007/978-3-319-94794-5_2 2, 3

[18] L. Gerlach, A. Ivliev, J. Méndez, S. Meusel, R. Dachselt, and M. Krötzsch. EvonNemo - a symbiosis of Datalog tracing and proof tree visualization. In *Int. Workshop Explainable Logic-Based Knowl. Repres.*, 11 2024. https://imld.de/cnt/uploads/2024-XLoKR-EvonNemo.pdf. 1, 3

[19] J. R. Haritsa. The picasso database query optimizer visualizer. *Proc. VLDB Endow.*, 3(1-2):1517–1520, 9 2010. doi: 10.14778/1920841.1921027 2

[20] A. Hinderks, M. Schrepp, and J. Thomaschewski. A benchmark for the short version of the user experience questionnaire. In *Proc. Int. Conf. Web Inf. Syst. Technol.*, pp. 373–377. INSTICC, SciTePress, 2018. doi: 10.5220/0007188303730377 3

[21] M. Horridge, R. S. Gonçalves, C. I. Nyulas, T. Tudorache, and M. A. Musen. Webprotégé: A cloud-based ontology editor. In *Companion Proc. WWW*, pp. 686–689. ACM, 2019. doi: 10.1145/3308560.3317707 2

[22] A. Ivliev, L. Gerlach, S. Meusel, J. Steinberg, and M. Krötzsch. Nemo: Your friendly and versatile rule reasoning toolkit. In *Proc. Int. Conf. Princ. Knowl. Represent. Reason.*, pp. 743–754. IJCAI, 2024. doi: 10.24963/kr.2024/70 1, 2

[23] W. Javed and N. Elmqvist. Exploring the design space of composite visualization. In *IEEE Pac. Vis. Symp.*, vol. 5, pp. 1–8, 2012. doi: 10.1109/PacificVis.2012.6183556 2

[24] H. Jordan, B. Scholz, and P. Subotic. Soufflé: On synthesis of program analyzers. In *Comput. Aided Verif.*, vol. 9780 of *LNCS*, pp. 422–430. Springer, 2016. doi: 10.1007/978-3-319-41540-6_23 1, 2

[25] M. Krötzsch. Modern Datalog: Concepts, methods, applications. In *Proc. RW 2024 & RW 2025*, vol. 138 of *OASIcs*. Dagstuhl Publishing, 2025. doi: 10.4230/OASIcs.RW.2024/2025.7 1

[26] Lain. OBS Studio: Open Broadcast Software. https://obsproject.com/, 2026. Accessed: 06-2026. 3

[27] S. Lieber, B. De Meester, P. Heyvaert, F. Brückmann, R. Wambacq, E. Mannens, R. Verborgh, and A. Dimou. Visual notations for viewing rdf constraints with unshacled. *Semantic Web*, pp. 1–36, 11 2021. doi: 10.3233/SW-210450 2

[28] D. Maier, K. T. Tekle, M. Kifer, and D. S. Warren. Datalog: concepts, history, and outlook. In *Declarative Logic Programming: Theory, Systems, and Applications*, vol. 20 of *ACM Books*, pp. 3–100. ACM / Morgan & Claypool, 2018. doi: 10.1145/3191315.3191317 1

[29] J. Méndez, C. Alrabbaa, P. Koopmann, R. Langner, F. Baader, and R. Dachselt. Evonne: A visual tool for explaining reasoning with OWL ontologies and supporting interactive debugging. *Comput. Graph. Forum.*, 3 2023. doi: 10.1111/cgf.14730 2

[30] J. Méndez, L. Gerlach, A. Ivliev, M. Krötzsch, and R. Dachselt. Nemo+NEV learning environment study. Qualitative Preregistration, OSF Registries, 12 2025. doi: 10.17605/OSF.IO/6KHY9 3

[31] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. RDFox: A highly-scalable RDF store. In *ISWC*, vol. 9367 of *LNCS*, pp. 3–20. Springer, 2015. doi: 10.1007/978-3-319-25010-6_1 2

[32] T. Petry. Visual EXPLAIN for MySQL. https://mysqlexplain.com/, 2026. Accessed: 06-2026. 2

[33] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever. Robust speech recognition via large-scale weak supervision. In *Proc. Int. Conf. Mach. Learn.* JMLR, 2023. https://proceedings.mlr.press/v202/radford23a/radford23a.pdf. 3

[34] J. C. Roberts. State of the Art: Coordinated Multiple Views in Exploratory Visualization. In *Proc. Int. Conf. Coordinated Mult. Views Explor. Vis.*, pp. 61–71, 7 2007. doi: 10.1109/CMV.2007.20 2

[35] J. Sauro and J. S. Dumas. Comparison of three one-question, post-task usability questionnaires. In *Proc. ACM CHI*, pp. 1599–1608. ACM, 2009. doi: 10.1145/1518701.1518946 3

[36] B. L. Sven Köhler and and Y. Smaragdakis. Declarative datalog debugging for mere mortals. In *Datalog in Academia and Industry - 2nd Int. Workshop*, vol. 7494 of *LNCS*, pp. 111–122. Springer, 2012. doi: 10.1007/978-3-642-32925-8_12 1

[37] S. Swaid and T. Suid. Computational thinking education: Who let the dog out? In *Int. Conf. Comput. Sci. Comput. Intell.*, vol. 6, pp. 788–792, 2019. doi: 10.1109/CSCI49370.2019.00150 2

[38] C. A. Wieland. Two explanation facilities for the deductive database management system DeDEx. In *Proc. Int. Conf. Entity-Relationship Approach*, pp. 189–203. ER Institute, 1990. 1, 2

[39] C. Williams. Buzz. https://chidiwilliams.github.io/buzz/docs/, 2026. Accessed: 06-2026. 3

[40] J. M. Wing. Computational thinking. *Commun. ACM.*, 49(3):33–35, 3 2006. doi: 10.1145/1118178.1118215 2, 3