Boosting Vampire with Neural Networks*

Martin Suda[1], Filip Bártek[1]

[1]Czech Technical University in Prague, Czech Republic

Abstract

In recent years, one of the research interests in our group has been enhancing the performance of automatic theorem provers using neural networks. This approach involves identifying heuristic choice points within a prover and replacing traditional implementations with neural models trained—based on prior prover experience—to make more effective decisions. Often, ingenuity is needed to define a machine learning task whose optimization will lead to the ultimate goal of solving more problems by the prover. In this paper, we share several insights and lessons learned from developing efficient neural guidance for the automatic theorem prover Vampire.

1. Introduction

Vampire is an automatic theorem prover (ATP) for first-order logic and beyond [1]. The difficulty of the theorem proving task—which is only semi-decidable for first-order logic—mandates that a competitive system, such as Vampire, must combine state-of-the-art calculi, optimized engineering, and strong heuristics. While a calculus implicitly defines the search space for a proof, heuristics helps to shape this space and to navigate it as efficiently as possible.

Recent years have seen a surge of interest in applying Machine Learning (ML) techniques to automatically evolve heuristics for various logic-based solvers, to guide proof search, and optimize algorithm portfolios [2]. In this paper, the authors survey their own modest contribution to this emerging field, using Vampire as the target solver and deep neural networks as the core ML technology.

ATP preliminaries. Vampire can be described as a refutational, saturation-based theorem prover implementing the superposition calculus, sharing this category with ATPs such as E [3], iProver [4], or SPASS [5].

By refutational we mean that Vampire seeks a proof by contradiction: in order to show that a conjecture G logically follows from a set of axioms Ax, the conjecture is negated and the system then works on the equivalent the task of showing that the set of formulas $Ax \cup \{\neg G\}$ is unsatisfiable. Typical is also the reliance formulas of simple shape called *clauses*. During preprocessing, the input gets efficiently translated into an equisatisfiable set of clauses, i.e., to a normal form called the Clause (or Conjuctive) Normal Form (CNF).

Superposition is a state-of-the-art deduction calculus with a built-in support for equality reasoning [6]. A calculus can be understood as an abstract algorithm defining how to derive new logical facts from those already established. Here, in the refutational context, the ultimate goal is to derive the empty clause, which serves as an atomic witness of unsatisfiablity.

Saturation is a way of organizing the search for the empty clause. It is, *a priory*, an exhaustive process (every derivable fact should be considered eventually), but incorporates tricks to prevent or at least postpone the imminent combinatorial explosion (prominent example being redundancy elimination). Saturation-based ATPs typically use two sets of clauses, here called the *active* and the *passive* set, and the process evolves around iteratively promoting (also called "activating") a clause from the passive set to the active set and performing all the inferences of the calculus between the promoted clause and all

10th International Workshop on Satisfiability Checking and Symbolic Computation, August 2, 2025, Stuttgart, Germany

*The work was supported by the Czech Science Foundation project 24-12759S.

martin.suda@cvut.cz (M. Suda)

0000-0003-0989-5800 (M. Suda); 0000-0002-1822-2651 (F. Bártek)

the active ones. The moment of choosing the next passive clause to promote is called *clause selection* and is the main heuristic target in an APT.

Neural Toolkit. For the following, we assume the reader to be familiar with the basic neural network terminology. The systems we describe make use of multi-layer perceptrons (MLPs), i.e., the basic building blocks consisting of matrix multiplications, vector additions, and component-wise non-linearities, Recursive Neural Networks (RvNNs) [7], and Graph Neural Networks (GNNs) [8, 9].

2. Neurally Boosted Vampires

We now survey four systems, each extending Vampire with a neural architecture to improve the performance of a heuristic choice point in the prover. The systems were evaluated on one or more standard problem libraries in the field, including the TPTP library [10], SMT-LIB [11], and the Mizar40 [12] export of the Mizar mathematical library [13] (referred below simply as Mizar).

2.1. Deepire 1.0

The Deepire system targets for improvement the clause selection heuristic, probably the most important choice point in a saturation-based theorem prover [14]. (Its significance is best appreciated if we consider an imaginary guiding network able to select exactly the clauses appearing in a sought proof. Using such guidance would clearly minimize the overhead connected with producing clauses that will never be useful, which typically constitute a majority.)

The main distinguishing feature of Deepire 1.0 [15, 16] is that it purposefully ignores logical content of the clause and only uses the clause's derivation history as a basis for discrimination. In other words, the guiding network is not allowed to look at "what a clause says", but only at "where a clause is coming from".

The derivation history is a directed acyclic graph. It grows from the nodes of the input clauses, each marked with an axiom it originates from, and its internal nodes stand for the derived clauses, each marked with the used derivation rule and references to its parents. Deepire unfolds an RvNN along the derivation history, embedding each of its nodes into a vector space. A small MLP is then used to turn such embedding into an evaluation of the corresponding clause. An important benefit of this approach is its low computational overhead: adding a new clause to the graph is a constant time operation.

The organization of the training process is taken from ENIGMA [17]. From each successful proof attempt of an unguided prover, selected clauses are recorded and split into positive, those that appeared in the discovered proof, and negative, the rest. A binary classifier is then trained to recognize the positively labeled clauses in the future. With its focus on selected clauses, this ENGIMA-style training strives to improve the existing clause selection heuristic.

There are many ways in which the learned advice can be integrated into the prover. Deepire 1.0 [15] explores adding a new queue, either sorted by the raw logits produced by the evaluation MLP, or having them grouped there, first the positive then negative ones, and using a tie-breaking criterion such clause age in each group. Yet another option, which turned out to be the most effective, is a layered clause-selection scheme [18], which selects clauses from the positive and the negative groups under some ratio, but uses the original clause selection heuristic within each group.

Deepire 1.0 was able to boost Vampire's performance in terms of the number of solved problems by $41\,\%$ on a relevant subset of SMT-LIB and by $28\,\%$ (resp. by $43\,\%$ after several rounds of looping, c.f. [19]) on Mizar. These evaluations use a fixed time limit per problem and thus the overhead from evaluating the guiding network (reaching up to $40\,\%$ of the allotted time) had to be compensated by correspondingly better clause selection choices.

2.2. Neural Precedence Recommender

The heuristic choice point targeted by the Neural Precedence Recommender [20] is the term ordering parametrizing the superposition calculus. Vampire predominantly uses the Knuth-Bendix Ordering (KBO) [21], which is determined by a *precedence*, a total ordering on the signature symbols.

The ordering (together with a literal selection function [22]) *constrains* the inference rules of the calculus in the sense that not every concrete inference that fits the rule's general template (and could be in principle allowed, as it is sound) is required to be performed by the prover (as there is a completeness argument that does not need it). Thus, inferences violating the ordering constraints can be skipped, which helps the prover fight the inherent combinatorial explosion.

Different precedences may have vastly different, but usually hard-to-predict, effects on proof search. Provers typically provide some rudimentary precedence construction schemes, like "sort symbols by arity" or "sort symbols by the number of occurrences in the input", but it is hard to pick the best one in advance. The Neural Precedence Recommender (NPR) aims to overcome this limitation.

NPR uses a GNN to analyze the input problem and output a precedence that will likely lead to a quick search for a proof. What this boils down to during inference is that the GNN computes for every symbol a cost and then recommends a precedence obtained by sorting the symbols by this cost.

To train the GNN, we collect prover experiences over a set of problems from the TPTP library [10] and randomly sampled pairs of precedences. Precedence π is considered better than ρ on problem P, if Vampire was able to solve P under fewer iterations of the saturation loop when using π than when using ρ. To bridge the gap between discriminating precedences and the symbols, the key trick is to promote the symbol costs c to a precedence cost C via

$$C(\pi) = \sum_{i=1}^{n} i \cdot c(\pi_i),$$

which has the desired property that C-minimizing precedences are exactly the ones that sorts the symbols by their cost. We then use the standard binary cross-entropy loss formula, effectively pushing the cost difference $C(\rho) - C(\pi)$ to be large for any training pair (π, ρ) in which π is considered better than ρ.

Once selected, a precedence must be fixed during the whole proof attempt. Thus only one GNN invocation is needed to make use of the recommender. We found the trained network to outperform a state-of-the-art precedence construction scheme by more than $4\,\%$ on unseen problems.

2.3. Clause Weight Recommender

Similarly to systems like ENIGMA or Deepire, the Clause Weight Recommender (CWR) [23] targets the clause selection heuristic. However, to stay as lightweight as possible in terms of evaluation speed, its design hard-wires a strong assumption about how clause's quality should be measured. It generalizes the standard clause weight criterion, which boils down to counting symbol occurrences, and allows each symbol to be given a distinct weight, so that the (generalized) weight of a clause arises as the sum of the weights of the symbols occurring in it.

The task of choosing symbol weights suitable for solving a given problem is delegated to a GNN, which CWR invokes at the start of the theorem proving attempt in a one-off manner. The GNN architecture is very similar to that of NPR. It consists of nodes corresponding to signature symbols, variables, terms, literals, and clauses and of edges of various kinds, such as the incidence relation between a clause and one of its literals or the relation between an atom and its predicate symbol. The graph structure then works as a scaffolding for several message-passing rounds, in which the nodes exchange messages computed from their embeddings and influenced by the edge kinds. A small MLP is then used to compute symbol weights from the final embeddings of the symbol nodes.

Training data is collected from successful prover runs and the training methodology combines elements from NPR and ENIGMA. As with ENIGMA, proof clauses are labeled as positive while the remaining selected are considered negative. As with NPR, we arrange these into pairs (C^+, C^-), each

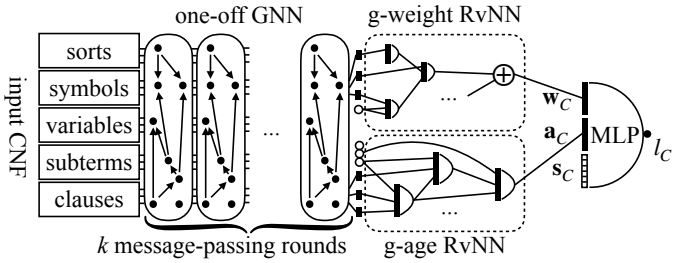

Figure 1: Clause evaluation neural architecture of Deepire 2.0.

consisting of positive and negative clause,[1] and strive to push the predicted probability of C^+ being preferable to C^-, defined as

$$p(C^+, C^-) = \text{sigmoid}(W(C^-) - W(C^+)),$$

towards 1 using the standard cross-entropy loss.

It is interesting to note we had to partially restrict the way in which the MLP produces a symbol weight from the symbol embedding to only allow for strictly positive symbol weights. (We achieved this by passing the MLP result through a final non-linearity $1 + \log(1 + e^x)$.) Without this restriction, negative weight symbols could quickly lead to diverging prover runs involving infinite derivation chains of increasingly long clauses with decreasing negative weights. In other words, insisting on strictly positive symbol weights was confirmed as a good measure to maintain prover fairness in practice.

On the first-order fragment of TPTP, when compared to uniformly weighting symbols, CWR was able to improve Vampire's performance by $6.6\,\%$ and by $2.1\,\%$ compared to a goal-directed variant of the uniformly weighting strategy.

Thanks to the simplicity of the artifact produced by the GNN, namely a single vector of symbol weights, we were able to inspect what the network deems relevant for proving problems effectively. We observed several interesting patterns including: 1) small weights of conjecture-related symbols, prioritizing goal directed search, 2) bumped-up "Tseitin variables" preventing saturation from undoing compact clausification quickly, or 3) low weight of the variable place-holder, suggesting a preference for general clauses over specific instances.

2.4. Deepire 2.0

As the name suggests, the Deepire 2.0 system [24] is a successor of Deepire 1.0 and as such it targets the clause selection heuristic for improvement.

One of the distinguishing features of version 2.0 is a much more sophisticated neural network architecture. The architecture (see Fig. 1) consists of a GNN invoked on the input problem's clause normal form (similarly to NPR and CWR, the GNN is run only once, before saturation begins), and two independent RvNNs, one evolving along the clause derivation history (similarly to Deepire 1.0, but using input clause embeddings provided by the GNN) and the other along the syntactic tree of the clauses (similarly to ENIMGA-NG [25], but using symbol embeddings from the GNN). We can think of the two RvNNs as generalizing and possibly improving upon the two most commonly used standard clause evaluation functions, clause age and clause weight (also generalized, although in a different way, by CWR). Embeddings coming from the two RvNNs, together with a small set of simple features, are turned into clause's logit in a final MLP.

Another important improvement is that Deepire 2.0 abandons ENGIMA-style learning, aiming to be more precise in how successful prover runs are turned into training data. We termed the new approach *RL-inspired learning operator*, because its construction follows from the principles of reinforcement learning. The gist of this improvement (cf. also *classic* vs *dynamic* data of [26]) is to see successful

[1] Either all such pairs can be generated, or a suitably large subset sampled randomly.

prover runs as *traces*, each trace being a sequence of snapshots of the passive clause set at the clause selection moments. Each such moment then serves as an independent opportunity to learn from. This is in contrast with the ENGIMA-style, which creates only a single binary classification task per prover run. In a nutshell, the RL-inspired approach is more faithful to the view in which an agent interacts with the theorem proving environment and picks clause selection actions towards the goal of proving a theorem as quickly as possible. This is also reflected in how we integrate the learned advice: Deepire 2.0 uses a single clause selection queue ordered by the clause's logits.

In an experiment on TPTP in a real time evaluation, Deepire 2.0 was found to improve over a baseline strategy, from which it initially learns, by impressive 20 %.[2] This can be compared to the much more modest result of CWR, where however, the time used to evaluate its GNN was not included in the time limit. More generally, note that previous works on ML-guided clause selection were rarely evaluated on TPTP due to its diverse nature, which makes an improvement much harder to achieve. For instance, Deepire 1.0 would not even be eligible there, as there are no axioms common across the whole library.

3. Final Thoughts

There are many don't-care non-deterministic choice points in an ATP and they are traditionally governed by hand-crafted heuristics. These heuristics can often be replaced by a neural model and trained in various ways from previous prover experiences to help the prover make favorable decisions. When designing the corresponding neural architecture, it is important to deliver (and translate to the "language of neurons") the necessary context relevant for deciding well. This may involve feature engineering and other forms of processing. A good balance must be struck between faithful representations that may take longer to process and easy-to-compute abstractions that can be processed more efficiently. It may not pay off, from the perspective of the ultimate prover performance, to buy an epsilon of additional decision quality for too big of a delta in processing time.

We have seen several ways in which prover experience can be turned into training data for the integrated network. Aligning the arising machine-learning *proxy* task with the ultimate *target* of "proving more theorems" is an ongoing research topic. Target improvements can be achieved already with simple proxies based on supervised learning, although reinforcement learning appears to lend more flexibility.

In all the four systems reviewed in this survey the source of knowledge came from the contrast between proof clauses versus the rest, i.e., learning from successes. It is an open question whether we could analogously make a good use of the information about failures. Another arising research topic is interpretability of the learned advice. We have had a limited success in this regard with our approaches, namely with the Clause Weight Recommender, which produces a relatively simple artifact that can be inspected. By default, however, deep neural networks are opaque, resistant to reverse engineering of what got learned.

Declaration on Generative AI

No generative AI was used.

References

[1] F. Bártek, A. Bhayat, R. Coutelier, M. Hajdú, M. Hetzenberger, P. Hozzová, L. Kovács, J. Rath, M. Rawson, G. Reger, M. Suda, J. Schoisswohl, A. Voronkov, The Vampire diary, in: R. Piskac, Z. Rakamaric (Eds.), Computer Aided Verification - 37th International Conference, CAV 2025,

[2] Additionally, Deepire 2.0 solved 130 TPTP problems of rating 1.0, i.e., problems not solved by any known system in the last official library-wide evaluation.

Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III, volume 15933 of *LNCS*, Springer, 2025, pp. 57–71.

[2] V. Ganesh, S. Szeider, Machine learning for solvers and provers (ML4SP) invited talks workshop, https://ml4sp.github.io/, 2025. Accessed: 2025-10-31.

[3] S. Schulz, S. Cruanes, P. Vukmirovic, Faster, higher, stronger: E 2.3, in: P. Fontaine (Ed.), Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings, volume 11716 of *LNCS*, Springer, 2019, pp. 495–507.

[4] A. Duarte, K. Korovin, Implementing superposition in iProver (system description), in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II, volume 12167 of *LNCS*, Springer, 2020, pp. 388–397.

[5] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, P. Wischnewski, SPASS version 3.5, in: R. A. Schmidt (Ed.), Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings, volume 5663 of *LNCS*, Springer, 2009, pp. 140–145.

[6] L. Bachmair, H. Ganzinger, Rewrite-based equational theorem proving with selection and simplification, J. Log. Comput. 4 (1994) 217–247.

[7] A. Küchler, C. Goller, Inductive learning in symbolic domains using structure-driven recurrent neural networks, in: KI 1996, volume 1137 of *LNCS*, Springer, 1996, pp. 183–197.

[8] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks 20 (2009) 61–80.

[10] G. Sutcliffe, The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0, Journal of Automated Reasoning 59 (2017) 483–502.

[11] C. Barrett, P. Fontaine, C. Tinelli, The Satisfiability Modulo Theories Library (SMT-LIB), www.SMT-LIB.org, 2016.

[12] C. Kaliszyk, J. Urban, MizAR 40 for Mizar 40, J. Autom. Reason. 55 (2015) 245–256.

[13] A. Grabowski, A. Kornilowicz, A. Naumowicz, Mizar in a nutshell, J. Formaliz. Reason. 3 (2010) 153–245.

[14] S. Schulz, M. Möhrmann, Performance of clause selection heuristics for saturation-based theorem proving, in: N. Olivetti, A. Tiwari (Eds.), Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings, volume 9706 of *LNCS*, Springer, 2016, pp. 330–345.

[15] M. Suda, Improving ENIGMA-style clause selection while learning from history, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *LNCS*, Springer, 2021, pp. 543–561.

[16] M. Suda, Vampire with a brain is a good ITP hammer, in: B. Konev, G. Reger (Eds.), Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings, volume 12941 of *LNCS*, Springer, 2021, pp. 192–209.

[17] J. Jakubuv, J. Urban, ENIGMA: efficient learning-based inference guiding machine, in: H. Geuvers, M. England, O. Hasan, F. Rabe, O. Teschke (Eds.), Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings, volume 10383 of *LNCS*, Springer, 2017, pp. 292–302.

[18] B. Gleiss, M. Suda, Layered clause selection for saturation-based theorem proving, in: P. Fontaine, K. Korovin, I. S. Kotsireas, P. Rümmer, S. Tourret (Eds.), Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual), volume 2752 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 34–52.

[19] J. Jakubuv, J. Urban, Hammering Mizar by learning clause guidance (short paper), in: J. Harrison, J. O'Leary, A. Tolmach (Eds.), 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA, volume 141 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 34:1–34:8.

[20] F. Bártek, M. Suda, Neural precedence recommender, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *LNCS*, Springer, 2021, pp. 525–542.

[21] D. E. Knuth, P. B. Bendix, Simple word problems in universal algebras, in: J. Leech (Ed.), Computational Problems in Abstract Algebra, Pergamon, 1970, pp. 263–297.

[22] K. Hoder, G. Reger, M. Suda, A. Voronkov, Selecting the selection, in: N. Olivetti, A. Tiwari (Eds.), Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings, volume 9706 of *LNCS*, Springer, 2016, pp. 313–329.

[23] F. Bártek, M. Suda, How much should this symbol weigh? A GNN-advised clause selection, in: R. Piskac, A. Voronkov (Eds.), LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023, volume 94 of *EPiC Series in Computing*, EasyChair, 2023, pp. 96–111.

[24] M. Suda, Efficient neural clause-selection reinforcement, in: C. W. Barrett, U. Waldmann (Eds.), Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings, volume 15943 of *LNCS*, Springer, 2025, pp. 403–422.

[25] K. Chvalovský, J. Jakubuv, M. Suda, J. Urban, ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E, in: P. Fontaine (Ed.), Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings, volume 11716 of *LNCS*, Springer, 2019, pp. 197–215.

[26] K. Chvalovský, K. Korovin, J. Piepenbrock, J. Urban, Guiding an instantiation prover with graph neural networks, in: R. Piskac, A. Voronkov (Eds.), LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023, volume 94 of *EPiC Series in Computing*, EasyChair, 2023, pp. 112–123.