MizAR 60 for Mizar 50

Jan Jakubův ✉ 📧
Czech Technical University in Prague,
Czech Republic

Karel Chvalovský ✉ 📧
Czech Technical University in Prague,
Czech Republic

Zarathustra Goertzel 📧
Czech Technical University in Prague,
Czech Republic

Cezary Kaliszyk ✉ 📧
Universität Innsbruck, Austria
INDRC, Prague, Czech Republic

Mirek Olšák 📧
Institut des Hautes Études Scientifiques,
Paris, France

Bartosz Piotrowski ✉ 📧
Czech Technical University in Prague,
Czech Republic

Stephan Schulz 📧
DHBW Stuttgart, Germany

Martin Suda ✉ 📧
Czech Technical University in Prague,
Czech Republic

Josef Urban 📧
Czech Technical University in Prague,
Czech Republic

Abstract

As a present to Mizar on its 50th anniversary, we develop an AI/TP system that automatically proves about 60 % of the Mizar theorems in the hammer setting. We also automatically prove 75 % of the Mizar theorems when the automated provers are helped by using only the premises used in the human-written Mizar proofs. We describe the methods and large-scale experiments leading to these results. This includes in particular the E and Vampire provers, their ENIGMA and Deepire learning modifications, a number of learning-based premise selection methods, and the incremental loop that interleaves growing a corpus of millions of ATP proofs with training increasingly strong AI/TP systems on them. We also present a selection of Mizar problems that were proved automatically.

1   Introduction: Mizar, MML, Hammers and AITP

In recent years, methods that combine machine learning (ML), artificial intelligence (AI) and automated theorem proving (ATP) [44] have been considerably developed, primarily targeting large libraries of formal mathematics developed by the ITP community. This ranges from *premise selection* methods [2] and *hammer* [7] systems to developing and training learning-based *internal guidance* of ATP systems such as E [47, 49] and Vampire [37] on the thousands to millions of problems extracted from the ITP libraries. Such large ITP corpora have further enabled research topics such as *automated strategy invention* [57] and *tactical guidance* [15], learning-based *conjecturing* [58], *autoformalization* [34, 61], and development of metasystems that combine learning and reasoning in various feedback loops [59].

Starting with the March 2003 release[1] of the MPTP system [54] and the first ML/TP and hammer experiments over it [55], the Mizar Mathematical Library [3, 4, 22] (MML) and its subsets have as of 2023 been used for twenty years for this research, making it perhaps the oldest and most researched AI/TP resource in the last two decades.

1.1   Contributions

The last large *Mizar40* evaluation [32] of the AI/TP methods over MML was done almost ten years ago, on the occasion of 40 years of Mizar. Since then, a number of strong methods have been developed in areas such as premise selection and internal guidance of ATPs. In this work, we therefore evaluate these methods in a way that can be compared to the Mizar40 evaluation, providing an overall picture of how far the field has moved. Our main results are:

1. Over 75 % of the Mizar toplevel lemmas can today be proved by AI/TP systems when the premises for the proof can be selected from the library either by a human or a machine. This should be compared to 56 % in Mizar40 achieved on the same version of the MML. Over 200 examples of the automatically obtained proofs are analyzed on our web page.[2]

2. 58.4 % of the Mizar toplevel lemmas can be proved today without any help from the users, i.e., in the large-theory (hammering) mode. This should be compared to about 40.6 % achieved on the same version of the MML in Mizar40. In both cases, this is done by a large portfolio of AI/TP methods which is limited to 420 s of CPU time.

3. Our strongest single AI/TP method alone now proves in 30 s 40 % of the lemmas in the hammering mode, i.e., reaching the same strength as the full 420 s portfolio in Mizar40.

4. Our strongest *single* AI/TP method now proves in 120 s 60 % of the toplevel lemmas in the human-premises (*bushy*) mode (Section 6.6), i.e., outperforming the union of *all* methods developed in Mizar40 (56 %).

5. We show that our strongest method transfers to a significantly newer version of the MML which contains a lot of new terminology and lemmas. In particular, on the new 13 370 theorems coming from the new 242 articles in MML version 1382, our strongest method outperforms standard E prover by 58.2 %, while this is only 56.1 % on the Mizar40 version of the library where we do the training and experiments. This is thanks to our development and use of *anonymous* [25] logic-aware ML methods that learn only from the structure of mathematical problems. This is unusual in today's machine learning which is dominated by large language models that typically struggle on new terminology.

---

[1] `http://mizar.uwb.edu.pl/forum/archive/0303/msg00004.html`
[2] `https://github.com/ai4reason/ATP_Proofs`

1.2 Overview of the Methods and Experiments

The central methods in this evaluation are internal guidance provided by the ENIGMA (and later also Deepire) system, and premise selection methods. We have also used several additional approaches such as many previously invented strategies and new methods for constructing their portfolios, efficient methods for large-scale training on millions of ATP proofs, methods that interleave multiple runs of ATPs with restarts on ML-based selection of the best inferred clauses (*leapfrogging*), and methods for minimizing the premises needed for the problems by decomposition into many ATP subproblems. These methods are described in Sections 3, 4, and 5, after introducing the MML in Section 2. Section 6 describes the large-scale evaluation and its final results, and Section 7 showcases the obtained proofs.

2 The Mizar Mathematical Library and the Mizar40 Corpus

Proof assistant systems are usually developed together with their respective proof libraries. This allows evaluating and showcasing the available functionality. In the case of Mizar [4], the developers have very early decided to focus on its library, the MML (Mizar Mathematical Library) [3]. This was done by establishing a dedicated library committee responsible for the evaluation of potential Mizar articles to be included, as well as for maintaining the library. As a result, the MML became one of the largest libraries of formalized mathematics today. It includes many results absent from those derived in other systems, such as lattices [5] and random-access Turing machines [36].

All the data gathered and evaluations performed in the paper (with the exception of version-transfer in Section 6.6) use the same Mizar library version as the previous large evaluation [32] and all subsequent evaluation papers. This allows us to rigorously compare the methods and evaluate the improvement. That version of the library, MML 1147, when exported to first-order logic using the MPTP export [56] corresponds to 57 897 theorems including the unnamed toplevel lemmas. For a rigorous evaluation in the hammering scenario, we will further split this dataset into several training and testing parts in Section 6.2.

3 ENIGMA: ATP Guidance and Related Technologies

ENIGMA [11, 18–21, 25, 27–29] stands for "*Efficient Learning Based Inference Guiding Machine*". It is the first learning-guided ATP that in 2019 achieved large improvements over state-of-the-art saturation ATPs [29], and the main ingredient of the work reported here. This section summarizes previously published research on ENIGMA and also the related methods that were used to undertake the large-scale experiments done here (Section 6).

3.1 Saturation Theorem Proving Meets Machine Learning

Saturation Provers.   State-of-the-art automated theorem provers, like E Prover [45] and Vampire [37], perform the search for a contradiction, first translating the input first-order logic problem into a refutationally equivalent set of clauses. Then the prover operates the proof search using the *given clause algorithm*. In this algorithm, the proof state is split into two subsets, the set P of *processed clauses*, and the set U of unprocessed clauses. Clauses in U are ordered by a heuristic evaluation function. In each iteration of the main loop, the (heuristically) best clause in U is picked. This *given clause g* is then simplified with respect to all clauses in P. If it is not redundant, it is used in turn to simplify all clauses in P. After that, all generating inferences between g and the remaining clauses in P are performed. Both
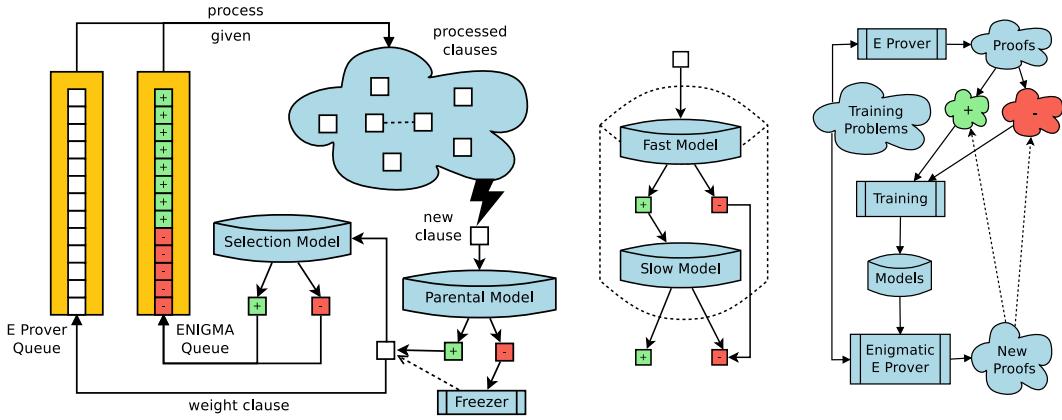
■ **Figure 1** Schema of E Prover with ENIGMA (left), of a two-phase selection model (middle), and of the prove-learn feedback loop (right).

the newly generated clauses and the simplified clauses from P are then completely simplified with respect to P, heuristically evaluated, and added to U. This process continues until the empty clause emerges (or until the system runs out of resources).

Training Data.   As of E 1.8 [48], E maintains an internal proof object [50] which allows it to inspect all proof clauses and designate all clauses that have been selected for processing and are part of the proof, as *positive training examples.* All clauses that have been selected for processing, but not contributed to the proof, are designated as *negative training examples.* Clauses that have not been processed at all are neither positive nor negative, reducing the total number of training examples to typically thousands of processed clauses, as opposed to millions of clauses generated. E allows the user to request the actual proof object, or to provide any combination of positive and negative training examples. Examples are provided in separate batches and are also annotated as positive or negative for easy processing.

ML-Based Selection.   Selection of the right given clause is critical in E, and an obvious point for the use of machine learning (ML). The positive and negative examples are extracted from previous successful proof searches, and a machine learning model is trained to score the generated clauses or to classify them as useful ($\boxplus$) or useless ($\boxminus$). E Prover selects the given clause from a priority queue, where the unprocessed clauses are sorted by various heuristics. ENIGMA extends E Prover with an additional queue where clauses positively classified by the ML model are prioritized. The ENIGMA queue is used together with the standard E selection mechanisms, typically in a cooperative way where roughly half of the clauses are selected by ENIGMA. This approach proved to be the most efficient in practice.

Parental Guidance.   Later ENIGMA [20] introduced learning-based *parental guidance*, which addresses the quadratic factor when doing *all* possible inferences among the processed clauses in classical saturation-based provers. Instead, an ML model is trained to prevent inferences between the parent clauses that are unlikely to meaningfully interact. When such an inference is recognized by the model as useless with a high degree of confidence, the child clause is not inserted into the set of unprocessed clauses U but its processing is postponed. To maintain completeness, the clause can not be directly discarded since the ML model might be mistaken. Instead, the clause is put into a "freezer" from which it can be retrieved in the case the

prover runs out of unprocessed clauses. As opposed to the above clause selection models, this method affects the standard E selection mechanism because the clause is not inserted into any queue. ENIGMA clause selection models and parental (*generation*) models can be successfully combined. This is schematically illustrated in Figure 1 (left).

Multi-Phase ENIGMA. ML-based multi-phase clause selection was introduced in [20] to deal with computationally expensive (*slow*) ML models, like graph neural networks (GNNs). In a *two-phase* selection model, a *faster* model is used for preliminary clause filtering, and only the clauses that pass are evaluated by the slower model. The fast model is expected to over-approximate on positive classes so that only clauses classified with high confidence as negatives are rejected. When parental guidance is added to the mix, this leads to a *three-phase* ENIGMA. This is schematically illustrated in Figure 1 (middle). *Aggressive forward subsumption* is an additional logic-complete pruning method based on efficient subsumption indexing in E [46]. We use it to eliminate many redundant generated clauses before calling more expensive ML methods (GNN) for clause evaluation. For the effect of such methods, see some of the 3-phase ENIGMA examples in Section 7.

Training. Strong ENIGMAs are typically developed in many prove-learn feedback loops [59] that proceed as follows. (1) The training data $\mathcal{T}$ are curated from (previous) successful proof searches. (2) A model $\mathcal{M}$ is trained on data $\mathcal{T}$ to distinguish positive from negative clauses. (3) The model $\mathcal{M}$ is run with the ATP (E), usually in *cooperation* with the strategy used to obtain the training data. Then we go to step (1) with the new data obtained in step (3). The loop, illustrated in Figure 1 (right), can be repeated as long as new problems are proved. We run this loop for several months in this work.

3.2 Gradient Boosted Decision Tree Classifiers and Features

ENIGMA supports classifiers based on Gradient Boosted Decision Trees (GBDTs). In particular, we experiment with XGBoost [8] and LightGBM [35]. Both frameworks are efficient and can handle large data well both in training and evaluation. For learning, we represent first-order clauses by numeric *feature vectors*. A decision tree is a binary tree with nodes labeled by conditions on the values of the feature vectors. Given a clause, the tree is navigated to the leaf where the clause evaluation is stored. Both frameworks work with a sequence (*ensemble*) of several trees, constructed in a progressive way (*boosting*). The frameworks differ in the underlying algorithm for the construction of decision trees. XGBoost constructs trees level-wise, while LightGBM leaf-wise. This implies that XGBoost trees are well-balanced. On the other hand, LightGBM can produce much deeper trees, and the tree depth limit is indeed an important learning meta-parameter that can be optimized.

ENIGMA extracts various syntactic information from a first-order clause and stores them in the feature vector of the clause. Given a finite set of features, each feature is assigned an index in the feature vector, and the corresponding feature value is stored at this index. For example, a typical clause feature is the clause length. ENIGMA supports the following. **Vertical Features** are constructed by traversing the clause syntax tree and collecting all top-down oriented symbol paths of length 3. Additionally, to abstract from variable names and to deal with possible collisions of Skolem symbols, all variables are replaced by a special name $\odot$ and all Skolem symbols by $\circledast$. **Horizontal Features** introduce for every term $f(t_1, \ldots, t_n)$, a new feature $f(s_1, \ldots, s_n)$, where s_i is the top-level symbol of t_i. **Count Features** include the clause length, literal counts, and similar statistics. **Conjecture Features** embed the conjecture to be proved in the feature vector. Thusly, ENIGMA is able to provide goal specific predictions. **Parent Features** represent a clause by features

(concatenated or summed) of its parents. **Feature Hashing** is an important step towards large data in ENIGMA [11]. It significantly reduces the feature vector size and thusly allows handling of larger data. Each feature is represented by a unique string identifier. This string is passed through the hashing function and the hash modulo the selected *hash base* is used as the feature index. **Symbol Anonymization** allows to abstract from specific symbol names [25]. During the extraction of clause features, all symbol names are replaced by symbol arities, keeping only the information whether the symbol is a function or a predicate. In this way, a decision tree classifier does not depend on symbol names, at the price of symbol collisions, which are however empirically mitigated by collecting longer paths as features.

3.3   Graph Neural Network (GNN) Classifiers

Anonymizing *graph neural networks* provide an alternative approach for abstracting from specific terminology. ENIGMA uses [25] a symbol-independent GNN architecture initially developed for guiding tableaux search [39] implemented in TensorFlow [1]. A set of clauses is directly represented by a hypergraph with three kinds of nodes for clauses, subterms/literals, and symbols. Relationships among the objects are represented by various graph edges, which allow the network to distinguish different symbols while abstracting from their names.

The GNN layers perform message passing across the edges, so the information at every node can get to its neighbors. This allows the network to see how the symbols are used without knowing their names. We always classify the new clauses together with the initial clauses which provide the context for the meaning of the anonymized symbols. During the ATP evaluation, predictions of hundreds of generated clauses are computed at once in larger batches, with the context given both by the initial and the processed clauses. The context can be either *fixed*, containing an initial segment of the initial and processed clauses, or it can be a *shifting context* using a window of clauses with the best GNN evaluation.

3.4   Additional Related Techniques

GPU Server Mode allows using GPUs for real-time evaluation [20]. To reduce the GPU overhead of model loading, we developed a Python GPU server, with preloaded models that can distribute the evaluation over several GPUs. E Prover clients communicate with the server via a network socket. We fully utilize our physical server[3] when we run 160 instances of E prover in parallel. Running both the server and clients on the same machine reduces the network communication overhead.

Leapfrogging addresses the problem of evolving context when new given clauses are selected [10]. We run ENIGMA with a given abstract limit and generate a larger set of clauses. Then we run a premise selection on these generated clauses (e.g., only processed clauses), take the good clauses, and use them as input for a new ENIGMA run. A related *split/merge* method involves repeatedly splitting the generated clauses into components that are run separately and then merged with premise selection. This is inspired by the idea that harder problems consist of components that benefit from such divide-and-conquer approaches.

Deepire is an extension [51, 52] of Vampire [37] by machine-learned clause selection guidance, generally following the ENIGMA-style methodology. It is distinguished by its use of recursive neural networks for classifying the generated clauses based solely on their derivation history. Thus Deepire does not attempt to read "what a clause says", but only

---

[3] 36 hyperthreading Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz cores, 755 GB of memory, and 4 NVIDIA GeForce GTX 1080 Ti GPUs.

bases its decisions on "where a clause is coming from". This allows the clause evaluation to be particularly fast, while still being able to recognize and promote useful clauses, especially in domains with distinguished axioms which reappear in many problems.

4 Learning Premise Selection From the MML

When an ATP is used over a large ITP library, typically only a small fraction of the facts are relevant for proving a new conjecture. Since giving too many redundant premises to the ATP significantly decreases the chances of proving the conjecture, premise selection is a critical task. The most efficient premise selection methods use *data-driven* or *machine-learning* approaches. If T is a set of theorems with their proofs and C is a set of conjectures without proofs, the task is to learn a (statistical) model from T, which for each conjecture $c \in C$ will rank (or select a subset of) its available premises according to their relevance for producing an ATP proof of c. Two main machine learning settings can be used. In **Multilabel classification**, premises used in the proofs are treated as opaque labels and a machine learning model is trained to label conjectures based on their features. **Binary classification** aims to recognize pairwise-relevance of the (*conjecture, premise*) pairs, i.e. to estimate the chance of a premise being relevant for proving the conjecture based on the features of both the conjecture and the premise.

The first setting is suitable for simpler, fast ML methods, like k-NN or Naive Bayes – these are described in Section 4.1. The second setting (Section 4.2) allows using more powerful ML architectures, like GBDTs and GNNs (Sections 3.3 and 3.2). However, this setting also requires selecting *negative examples* for training [41], which increases its complexity.

4.1 Multilabel Premise Selection ($\mathcal{K}$, $\mathcal{N}$, $\mathcal{R}$)

Naive Bayes and k-nearest neighbors were the strongest selection methods in the Mizar40 evaluation [32]. In this work, we improve them and apply them together with newer methods.

k-NN ($\mathcal{K}$). The k-nearest neighbours algorithm, when applied to premise selection, chooses k facts closest to the conjecture in the feature space and selects their dependencies. Already known modifications of the standard k-NN include considering the number of dependencies (proofs with more dependencies are longer and thus less important) and TF-IDF (rare features are more important) [30]. Additionally, we realize that we do not need to fix the k. Instead, we consider a small k and if the number of scored dependencies is too low, we increase the k and update the dependencies. This is repeated until the requested number of predictions is obtained. The k-NN-based predictions with fixed k will be denoted, e.g., by $\mathcal{K}_{512}$, while with variable k this will be $\mathcal{K}_{var}^{fea}$, where *fea* specifies the features used.

Naive Bayes ($\mathcal{N}$). The sparse Naive Bayes algorithm estimates the relevance of a fact F by the conditional probability of F being useful (estimated from past proof statistics) under the condition of the features being present in the conjecture (again estimated from statistics). We also consider *extended features* of F, i.e., features of F and features of facts proved using F. Together with premise selection-specific weights this improves on the basic Naive Bayes and has already been used in HolyHammer and Sledgehammer. A complete derivation of the algorithm is given in [6]. The Naive Bayes predictions will be denoted $\mathcal{N}_{fea}$.

These algorithms can be parametrized by more complex *features*. We considered: `cp` for constants and paths (Section 3.2) in the term graph, `sub` for subterms, `au` for anti-unification features [33], `eni` for online ENIGMA features discussed in Section 3.2 and `uni` for the union

of all above. Finally, these algorithms also support the *chronological* mode, which in the learning phase discards proofs that use facts introduced after the current conjecture in the Mizar canonical order (`MML.LAR`). This slightly weakens the algorithms, but is compatible with the previous Mizar40 premise selection evaluation [32]. These will be marked by ^{chrono}.

Dependent Selection with RNNs ($\mathcal{R}$). Premise selection methods were originally mainly based on *ranking* the facts *independently* with respect to the conjecture. The highest ranked facts are then used as axioms and given to the ATP systems together with the conjecture. Such approaches (used also with GBDTs), although useful and successful, do not take into account that the premises are *not* independent of each other. Some premises complement each other better when proving a particular conjecture, while some highly-ranked premises might be just minor variants of one another. Recurrent neural network (RNN) encoder-decoder models [9] and transformers [60] (language models) turn out to be suitable ML architectures for modeling such implicit dependencies. Such models have been traditionally developed for natural language processing, however, recently they are also increasingly used in symbolic reasoning tasks [12, 16, 38, 43, 61], including premise selection [42].

4.2 Premise Selection as Binary Classification ($\mathcal{L}$, $\mathcal{G}$)

Gradient Boosted Decision Trees ($\mathcal{L}$). We use GBDTs (LightGBM) also for premise selection in the binary mode. They are faster to train than the deep learning methods, perform well with unbalanced training sets, and handle well sparse features. We fix the LightGBM hyperparameters here based on our previous experiments with applying GBDTs to premise selection [41]. In the binary setting, the GBDT scores the pairwise relevance of the conjecture and a candidate premise. Because the number of possible candidates is large (all preceding facts in the large ITP library), we first use the cheaper k-nearest neighbors algorithm to pre-filter the available premises. The predictions from LightGBM will be denoted as $\mathcal{L}$ below.

Dependent Selection with GNNs ($\mathcal{G}$). The message-passing GNN architecture described in Section 3.3 can also be applied to premise selection. Like RNNs, it can also take into account the dependencies between premises. As the GNN is relatively slow, we will use it in combination with a simpler premise selection method, such as k-NN, preselecting 512 facts. We will denote GNN predictions by $\mathcal{G}$ below. Both $\mathcal{L}$ and $\mathcal{G}$, can be indexed with the threshold on the score (like $\mathcal{L}_{0.1}$ or $\mathcal{G}_{-1}$), used to differentiate useful and useless clauses.

4.3 Ensemble Methods for Premise Selection ($\mathcal{E}$)

There are several ways how we can combine the premise selection methods discussed in previous subsections. Naturally, using different methods for different strategies works well, however, we also found that combining the predictions obtained from several methods and using them for a single prover run gives good and complementary results. Since prediction scores resulting from different algorithms are often incomparable [40], we only use the rankings produced by the various methods and based on this we create a combined ranking. We have compared several ways to combine rankings in previous work [30] and found that several averages work well: arithmetic mean, minimum, and geometric mean, with the harmonic mean giving experimentally the best results. Additionally, we add weights to the different combined methods. The weights give more priority to a stronger prediction method, but allow it to benefit from the simpler ones overall (by picking up some lost facts). Given

predictions from n different methods and method weights $w_1, \ldots, w_n$, assume that a fact has been ranked as r_1-th by the first method until and r_n-th by the last one. Then, the ensemble method would give that fact a score of $1/\sum_{i=1}^{n} \frac{w_i}{r_i}$. The scores of the facts obtained in this way are sorted, to get a ranking of all facts. The ensemble predictions will be denoted by $\mathcal{E}$, with methods and their weights in the super and subscript, for example $\mathcal{E}_{0.25,0.25,0.5}^{\mathcal{K},\mathcal{N},\mathcal{G}}$.

4.4 Subproblem Based Premise Minimization ($\mathcal{M}$)

The proof dependencies obtained by successful ATP runs typically perform better as data for premise selection than the dependencies from the human-written ITP proofs [7, 31]. However, some Mizar proofs are hundreds of lines long and it is so far unrealistic to raise the 75 % ATP performance obtained here in the bushy setting to a number close to 100 %. This means that if we used only ATP-based premise data, we would currently miss in the premise selection training 25 % of the proof dependency information available in the MML.

To remedy that, we newly use here *subproblem based premises*. The idea behind this is that a theorem with a longer Mizar proof consists of a series of natural deduction steps that typically have to be justified. Once ATP proofs of all such steps (we call them subproblems) for a given toplevel theorem are available, they can be used to prune the (overapproximated) set of human-written premises of the theorem. Such minimization also increases the chance of proving the theorem directly. In more detail, we consider the following approaches: (1) Use the premises from only ATP-proved subproblems, ignoring unproved subproblems. (2) Add to (1) all explicit Mizar premises of the theorem (possibly ignoring some background facts). (3) Add to (2) also the (semi-explicit) definitional expansions detected by the natural deduction module. (4) Add to (3) also some of the background premises, typically those ranked high by the trained premise selectors. When using (1) and (3), we were able to prove more than 1000 hard theorems (see Table 1 in Section 6.1). We also use (3) as additional proof dependencies for ATP-unproved theorems when training premise selectors (Section 6.2).

5 Strategies and Portfolios

Strategies. E, ENIGMA, Vampire and Deepire are parameterized by ATP strategies and their combinations. While ENIGMA-style guidance typically involves the application of a larger (neural, tree-based, etc.) and possibly slower statistical model to the clauses, standard ATP strategies typically consist of much faster clause evaluation functions and programs written in a DSL provided by the prover. Such programs can again be invented and learned in various ways for particular classes of problems. For the experiments here we have used many ATP strategies invented automatically by the BliStr/Tune systems [26, 27, 57]. They implement feedback loops that interleave targeted *parameter search* on problem clusters using engines like ParamILS [23], with a large-scale evaluation of the invented strategies used for evolving the problem clustering. Starting from few strategies, BliStr/Tune typically evolve each strategy on the problems where the strategy performs best. During our experiments with the systems we have developed several thousand E Prover strategies, many of them targeted to Mizar problems. Some of these are mentioned in the experiments in Section 6.

Robust Portfolios. Larger AI/TP systems and metasystems rely on portfolios [53] of complementary strategies that attack the problems serially or in parallel using a global time limit. In the presence of premise selection and multiple ATPs, such portfolios may consist of tens to hundreds of different methods. The larger the space of methods, the larger is the risk of overfitting the portfolio during its construction on a particular set of problems. For

example, naive construction of "optimal" portfolios by using SAT solvers for the set-cover problem (where each strategy covers some part of the solution space) often leads to portfolios that are highly specialized to the particular set of problems. This is mitigated in more robust methods such as the *greedy cover*, however, the overfitting there can still be significant. E.g., a 14-strategy greedy cover built in the Mizar40 experiments [32] solved 44.1 % of the random subset used for its construction, while it solved only 40.6 % of the whole MML, i.e., 8 % less.

To improve on this, we propose a more robust way of portfolio construction here, based again on the machine-learning ideas of controlling overfitting. Instead of simply constructing one greedy cover C (with a certain time budget) on the whole development set D and evaluating it in the holdout set H, we first split D randomly into two equal size halves D_1 and D_2. Then we construct a greedy cover C_1 only on D_1, and evaluate its performance also on D_2 and the full set D. This is repeated n times (we use $n = 1000$), which for large enough n typically guarantees that the greedy cover C_1^i will for some of the random splits D_1^i, D_2^i overfit very little (or even underfit). This can be further improved by evaluating the best (strongest and least overfitting) covers on many other random splits and selecting the most robust ones. We use this in Section 6 to build a portfolio that performs only 3.5 % worse on the (unseen) holdout set than on the development set used for its construction.

6   Experiments and Results

6.1   Bushy Experiments and Timeline

The final list of all 43 717 Mizar problems proved by ATPs in our evaluation is available on our web page.[4] The approximate timeline of the methods and the added solutions is shown in Table 1. This was continuously recorded on our web page,[5] which also gives an idea of how the experiments progressed and how increasingly hard problems were proved.

The large evaluation started in April 2020, as a follow-up to our work on ENIGMA Anonymous [25]. By combining the methods developed there and running with higher time limits, the number of problems proved by ENIGMA in the bushy setting reached 65.65 % in June 2020. This was continued by iterating the learning and proving in a large Malarea-style feedback loop. The growing body of proofs was continuously used for training the graph neural networks and gradient boosted guidance, which were used for further proof attempts, combined with different search parameters and later used also for training premise selection.

This included many grid searches on a small random subset of the problems over the thousands of differently trained GNNs and GBDTs corresponding to the training epochs, and then evaluating the strongest and most complementary ENIGMAs using the differently trained GNNs and GBDTs on all, or just *hard* (the so far ATP-unproved), problems. The total number of the saved snapshots of the GNNs corresponding to the training epochs and usable for the grid searches and full evaluations reached 15 920 by the end of the experiments in September 2021.[6] The longest GNN training we did involved 964 epochs and 12 days on a high-end NVIDIA V100 GPU card.[7] The GNN training occasionally (but rarely) diverged after hundreds of epochs, which we handled by restarts.

---

[4] `http://grid01.ciirc.cvut.cz/~mptp/00proved_20210902`
[5] `https://github.com/ai4reason/ATP_Proofs`
[6] For the grid searches, this was compounded by further parameters of the ENIGMA and E strategies.
[7] We generally use the same GNN hyper-parameters as in [25, 39] with the exception of the number of *layers* that varied here between 5 and 12, providing tradeoffs between the GNN's speed and precision.

Table 1 Timeline of the experiments. $\mathcal{B}$ are standard bushy premises, $\mathcal{M}$ are subproblem-minimized premises, $\mathcal{G}$, $\mathcal{L}$, and $\mathcal{K}$ are GNN/LightGDB/kNN-based premises, and $\mathcal{E}$ their ensembles.

solved	[%]	date	premises	methods/notes
38k	65.65	Jun 2020	$\mathcal{B}$	ENIGMA, reported on July 2nd at IJCAR'20[8]
40 268	69.57	Oct 2020	$\mathcal{B}$	ENIGMA
40 994	70.83	Nov 12	$\mathcal{M}$	ENIGMA, heuristic premise minimization
41 169	71.13	Nov 12	$\mathcal{M}$	Vampire with 300 s limit adds 175
41 792	72.20	Nov 27	$\mathcal{M}$	E/ENIGMA/Vampire with more premise minimization
42 206	72.92	Dec 7	$\mathcal{M}$	E/ENIGMA/Vampire with more premise minimization
42 471	73.38	Jan 6	$\mathcal{G}$, $\mathcal{E}$	E with BliStr/Tune strategies on $\mathcal{G}$, $\mathcal{E}$ premises
42 519	73.46	Jan 10	many	ENIGMA runs on all training predictions
42 826	73.99	May 14	$\mathcal{G},\mathcal{L},\mathcal{K}$	Vampire/Deepire runs – FroCoS'21 [52]
43 414	75.01	Jul 26	$\mathcal{M},\mathcal{B}$	2,3-phase ENIGMA, leapfrogging
43 524	75.20	Aug 21	$\mathcal{M}$	3-phase ENIGMA, shifting context, leapfrog., fwd subsump.
43 599	75.33	Aug 26	$\mathcal{L}$	3-phase ENIGMA, leapfrogging, fwd. subsumption
43 717	75.53	Sep 2	$\mathcal{M}$	mainly Vampire/Deepire

The total number of proofs that we trained the ENIGMA guidance on eventually reached more than three million, which in a pickled and compressed form take over 200 GB. Since the full data do not fit into the main memory of even large servers equipped for efficient GPU-based neural training, we have programmed custom pipelines that continuously load, mix and unload smaller chunks of data used for the ENIGMA training. For many problems, we obtained hundreds of different proofs, while for some problems we may have only a single proof. This motivated further experiments on how and with what frequency the different proofs should be represented in the training data. This was a part of the larger task of *training data normalization*, which included, e.g., removing or pruning very large proof searches in the training data that would cause memory-based GPU crashes.

The 75 % milestone was reached on July 26 2021[9] by using the freshly developed 2 and 3-phase ENIGMAs, together with differently parameterized leapfrogging (Section 3.4) runs. The strongest single 3-phase ENIGMA strategy has reached 56.4 % performance in 30 s on the bushy problems when trained and evaluated in a rigorous train/dev/holdout setting [20]. This best ENIGMA uses a parental threshold of 0.01, 2-phase threshold of 0.1, and context and query sizes of 768 and 256. Its (server-based) GNN has 10 layers trained on at most three proofs for each problem in the training set. See also Section 6.6 for its evaluation on a set of completely new 13 370 problems in 242 new articles of a later version of MML.

6.2 Training Data for Premise Selection

After several months of running the learning/proving loop in various ways on the problems, we used the collected data for training premise selection methods. In particular, at that point, there were 41 504 ATP-proved problems for which we typically had many alternative proofs and sets of premises, yielding 621 642 unique ATP proof dependencies. Since in the hammering scenarios we can also analyze the human-written proofs and learn from them, we have added for each ATP-unproved problem P its premises obtained by taking the union of

---

the ATP dependencies of all subproblems of P. In other words, we use subproblem-based premise minimization (Section 4.4) for the remaining hard problems. This adds 16 651 examples to the premise selection dataset. This dataset of 638 293 unique proof dependencies is then used in various ways for training and evaluating the premise selection methods on MML. In comparison with the Mizar40 experiments this is about six times more proof data. As usual in machine learning experiments, we also split the whole set of Mizar problems into the *train*ing, *devel*opment, and *holdout* subsets, using a 90 : 5 : 5 ratio. This yields 52 125 problems in the training set, 2896 in devel, and 2896 in the holdout set.

6.3   Training the Premise Selectors

We first train kNN and naive Bayes in multiple ways on the training subset using the different features (Section 4.1) and their combinations. For training the GNN and LightGBM, we first use kNN-based pre-selection to choose 512 most relevant premises for each problem. When training, we add for each example its positives (the real dependencies) and subtract them from the 512 premises pre-selected by kNN, thus forming the set of the negatives for the example. The GNN and LightGBM are thus trained to correct the mistakes done by kNN (a form of *boosting*). When predicting, this is done in the same way, i.e., first we use the trained kNN to preselect 512 premises which are then ranked by the GNN/LightGBM. We use both score thresholds (e.g., including all premises with score better than 0, -1 or -3), and fixed-sized slices as in other premise selection methods. With the same best version of ENIGMA, the strongest GNN-based predictor ($\mathcal{G}_{-1}$) solves 1089 problems compared to 870 solved when using the baseline kNN, which is a large (25.2 %) improvement. The GNN also outperforms LightGBM, which overfits more easily on the training data. Table 2 shows the performance on the devel and holdout sets of the main methods used in the evaluation.

6.4   ENIGMA Experiments on the Premise Selection Data

First, to train ENIGMA on the premise selection problems, we perform several prove/learn iterations with ENIGMA/GBDT on our premise slices. In loop (1), we start with three selected slices $\mathcal{G}_{-1}$, $\mathcal{L}_{0.1}$, and $\mathcal{K}_{64}$, which were found experimentally to be complementary. We evaluate strategy $\mathcal{S}_1$ (bls0f17) on the three slices obtaining 20 604 proved training problems. We train several decision tree (GBDT) models with various learning hyperparameters (tree leaves count, tree depth, ENIGMA features used). We use all the training proofs available. In loop (2), we evaluate several ENIGMA models trained on $\mathcal{B}$ (bushy problems) to obtain additional training data. After few training/evaluation iterations, the training data might start accumulating many proofs for some (easier) problems solved by many strategies. From loop (2) on, we, therefore, use only a limited number of proofs per problem. We either select randomly up to 6 proofs for each problem, or we select only specific proofs (e.g., the shortest, longest, and one medium-length proof). In loop (3), additional training data are added by ENIGMA/GNN runs on the premise slices, with GNN trained on the GBDT runs. In loop (4), we consider training data from 7 additional slices (variants of $\mathcal{G}$, $\mathcal{L}$, $\mathcal{K}$), obtained by running ENIGMA models trained of bushy problems. In loop (5), we extend the training data with bushy proofs of unsolved training problems obtained by our various previous efforts.

Starting from 1215 solved development problems, we ended up with 1735 problems solved after the fifth iteration. While we train GBDT models only on few selected slices, we evaluate the models on many more, up to 56, development slices covering all families $\mathcal{G}$, $\mathcal{L}$, $\mathcal{K}$, $\mathcal{E}$, and $\mathcal{N}$. We report the increasing number of training problems (*trains*) and the total of number of solved development problems by all the evaluated strategy/slice pairs (*devel* union). Since every strategy/slice pair is evaluated in 10 seconds, we construct the greedy cover of best 42

■ **Table 2** Machine learning evaluation of the premise selection models on the **D**evelopment and **H**oldout datasets. Note that the evaluation of GNN is presented here only for completeness, in practice we use it with a score-based threshold and fewer premises.

Model	100-Cover		100-Prec		Recall		AUC		Avg. Rank	
	D	H	D	H	D	H	D	H	D	H
$\mathcal{K}_{\mathrm{var}}^{\mathrm{cp}}$	83.3	82.3	8.837	8.713	386.8	401.9	92.03	91.27	90.17	97.98
$\mathcal{K}_{\mathrm{var}}^{\mathrm{au}}$	83.5	82.7	8.855	8.754	383.32	401.54	92.13	91.36	89.21	97.19
$\mathcal{K}_{\mathrm{var}}^{\mathrm{mi}}$	82.6	81.8	8.700	8.596	401.89	418.40	91.32	90.59	97.30	104.88
$\mathcal{K}_{\mathrm{var}}^{\mathrm{s0}}$	83.6	82.9	8.851	8.785	382.31	399.10	92.19	91.39	88.53	96.84
$\mathcal{N}_{\mathrm{cp}}$	87.8	87.0	9.739	9.665	300.49	310.72	94.77	94.32	62.64	67.51
$\mathcal{N}_{\mathrm{au}}$	88.0	87.5	9.748	9.714	298.66	307.82	94.84	94.44	61.99	66.31
$\mathcal{N}_{\mathrm{mi}}$	83.3	83.5	9.358	9.367	382.87	379.24	92.53	92.41	85.39	86.88
$\mathcal{N}_{\mathrm{s0}}$	88.3	87.5	9.776	9.720	299.10	308.67	94.85	94.41	61.85	66.60
$\mathcal{N}_{\mathrm{mi,chrono}}$	83.9	82.7	9.151	9.010	384.68	393.37	92.33	91.75	87.23	93.27
$\mathcal{L}$	82.9	83.1	9.077	9.090	410.06	408.06	91.53	91.26	95.11	97.74
$\mathcal{G}$	87.4	86.3	9.408	9.282	241.22	249.32	88.45	87.42	66.69	71.87
$\mathcal{E}_{.5,.5,\&\mathrm{avg}}^{\mathcal{N},\mathcal{K}}$	87.8	87.1	9.606	9.522	291.27	304.43	95.06	94.53	59.81	65.47
$\mathcal{E}_{.5,.5,\&\mathrm{geo}}^{\mathcal{N},\mathcal{K}}$	89.4	88.7	9.806	9.733	277.75	288.53	95.53	95.04	55.13	60.27
$\mathcal{E}_{.5,.5,\&\mathrm{har}}^{\mathcal{N},\mathcal{K}}$	89.4	88.9	9.822	9.780	276.34	286.23	95.53	95.08	55.06	59.94
$\mathcal{E}_{.5,.5,\&\mathrm{min}}^{\mathcal{N},\mathcal{K}}$	89.0	88.4	9.753	9.707	279.88	289.41	95.37	94.95	56.70	61.19
$\mathcal{E}_{.5,.25,.25}^{\mathcal{N},\mathcal{G},\mathcal{K}}$	92.1	91.1	10.237	10.160	228.79	248.16	96.64	96.20	44.06	48.76
$\mathcal{E}_{.5,.2,.2,.1}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}$	92.5	91.5	10.297	10.219	210.31	227.74	96.93	96.56	41.20	45.17
$\mathcal{E}_{.33,.33,.33}^{\mathcal{N},\mathcal{G},\mathcal{K}}$	91.3	90.4	10.091	10.014	261.10	272.85	96.20	95.71	48.51	53.64

strategy/slice pairs, to approximate the best possible result obtainable in 420 s (see columns *devel cover*). Since the development set has not been used in any way to train the GBDT models, we can see this as an approximation of the best possible result on the holdout set.

We reach 55.59 % of problems solvable in 420 s, only with ENIGMA/GBDT models. To compare this result to other methods, we construct compatible greedy covers for E Prover in *auto-schedule mode*, and for Vampire in *CASC mode*, that is, in their respective strongest default settings. We evaluate both provers on all 56 development slices, with 30 s limit per problem. For each prover, we construct a greedy cover of best 14 slices, again approximating the best possible result obtainable in 420 s. BliStr/Tune is our previously invented portfolio of 15 E strategies for Mizar bushy problems. We evaluate all 15 strategies on all 56 slices with 2 seconds per problem. Similarly, the greedy cover of length 210 is constructed. The column *pairs* specifies the greedy cover length considered in each case. The time limit for each strategy/slice pair is 420/*pairs*.

The training data obtained in five loops were finally used to train new ENIGMA/GNN models for premise selection slices. Various GNN models were trained (various numbers of layers, networks from various epochs) and evaluated with the limit of 5 s. As before, we construct the greedy cover of length 84 to simulate the best possible run in 420 s. ENIGMA/GNN performs even better then ENIGMA/GBDT, solving 57.66 % problems. The two ENIGMA/* portfolios cover together 1701 development problems (in 840 s), suggesting a decent complementarity of the methods. Note that only the ENIGMA/GBDT strategies can cover up to 1735 (see column *devel* on the left), which is 59.9 % of the development set.

■ **Table 3** Training of ENIGMA/GBDT models (left), and best covers of development set (right).

loop	trains	devel (union)	devel cover (in 420 s)	[%]
init	20 604	1215	-	-
(1)	25 240	1601	1516	52.33
(2)	25 725	1669	1555	53.69
(3)	25 887	1679	1560	53.88
(4)	29 266	1716	1591	54.94
(5)	37 053	1735	1610	55.59

prover (420 s)	cover	pairs	[%]
E 2.6 (auto-schedule)	1430	14	49.38
Vampire 4.0 (CASC)	1536	14	53.03
BliStr/Tune	1582	210	54.62
ENIGMA/GBDT	1610	42	55.59
ENIGMA/GNN	1670	84	57.66

Most of our ENIGMA models are combined with the baseline strategy bls0f17. This together with bls05fc are two strategies invented by BliStr/Tune [24] which perform well on premise selection data. We additionally use another two older BliStr [57] strategies mzr02 and mzr03 which perform well on bushy problems. We usually combine training data only from strategies with compatible term ordering and literal selection setting. However, data from strategies with incompatible orderings, were found useful when used in a reasonably small amounts. Few other BliStr and Vampire strategies, together with E in the *auto* mode, are used to gather additional solved development problems. With all our methods (ENIGMA & BliStr/Tune) and with additional Vampire runs of selected strategies, we have solved more than 62.7 % development problems. These results provide training data for the construction of the final holdout portfolio, as described in the next section.

6.5 Final Hammer Portfolio

With the large database of the development results of the systems run on the premise slices, we finally construct our ultimate hammering portfolio. For that, we use the *robust portfolio construction* method described in Section 5. In particular, we randomly split the development set into two equal-sized parts, and compute the 420 s greedy cover using our whole database of results on the first part. This greedy cover is evaluated on the second part, thus measuring the overfitting. This randomized procedure is repeated one thousand times. Then we (manually) select the 20 strongest and least overfitting portfolios and evaluate each of them on 80 more random splits, thus measuring how balanced they are on average. Typically, they reach up to 60.5 % performance on the whole devel set, so we choose a threshold of 59.5 % on the 160 random halves to measure the imbalance. The most balanced portfolio wins with 135 of the 160 random halves passing the threshold.

This final 420-second portfolio has 95 slices that solve 1749 (60.4 %) of the devel problems and 1690 (58.36 %) of the holdout problems. Table 4 shows the initial segment of 13 slices of this portfolio with the numbers of problems solved. The full portfolio is presented in Table 5. The first number t is the number of seconds to run the slice. The *base* column specifies the ATP strategy used, and *ENIGMA* describes what kinds of ENIGMA models are used (if any). We can see that GNN models dominate the schedule with fast runs. The schedule is closed by longer runs, notably also GBDT models, which while evaluated in a single-CPU setting, need several seconds to load the model. This means that we are favoring the GNN ENIGMAs thanks to the use of the preloaded GNN server, and a further improvement is likely if we also preload the GBDT models. Our single strongest GNN-based strategy solves 1178 of the holdout problems in 30 s using the $\mathcal{G}_{-1}$ predictions. This is 39.5 %, which is only 1.1 % less than the 40.6 % solved by the full 420 s portfolio constructed in the Mizar40 experiments.

■ **Table 4** The 13-slice prefix of the final portfolio of the 95 slices. Each column presents the premise selection method, the ATP method, and the number of problems solved up this slice cumulatively on the development and holdout sets. "V" stands for Vampire and "GNN" is ENIGMA/GNN model based on bls0f17. Moreover, $\mathcal{E}_{5221} = \mathcal{E}_{.5,.2,.2,.1}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}$ and $\mathcal{E}_{55} = \mathcal{E}_{.5,.5,\text{avg}}^{\mathcal{N},\mathcal{K}}$ and $\mathcal{E}_{533} = \mathcal{E}_{.5,.25,.25}^{\mathcal{N},\mathcal{G},\mathcal{K}}$.

$\mathcal{G}$	$\mathcal{E}_{5221}$	$\mathcal{L}$	$\mathcal{E}_{5221}$	$\mathcal{N}_{\text{uni}}$	$\mathcal{E}_{5221}$	$\mathcal{N}_{\text{eni}}$	$\mathcal{N}_{\text{eni}}$	$\mathcal{E}_{5221}$	$\mathcal{G}$	$\mathcal{E}_{55}$	$\mathcal{E}_{533}$	$\mathcal{E}_{533}$
GNN	GNN	GNN	GNN	V	V	GNN	GNN	GNN	GNN	V	V	GNN
984	1142	1215	1263	1297	1325	1346	1370	1381	1393	1405	1419	1444
1013	1157	1240	1275	1305	1321	1346	1364	1378	1386	1398	1407	1436

6.6 Transfer to MML 1382

In the final experiment, we run for 120 s the best trained ENIGMA (3-phase, see Section 6.1) on the bushy problems from a new version of Mizar (1382) that has 242 new articles and 13 370 theorems in them. ENIGMA not only never trained on any of these articles, but also never saw the new terminology introduced there. We also run the standard E auto-schedule for 120 s on the new version. ENIGMA proves 37 094 (52.7 %) of the 70 396 problems in the new library, while the E auto-schedule proves 24 158 (34.32 %) of them. ENIGMA thus improves over E by 53.55 % on the new library. We compare this with the old MML, where the trained ENIGMA solves 34 528 (59.65 %) of the 57 880 problems, and E solves 22 119 (38.22 %), i.e., the relative improvement there is 56.10 %.

Surprisingly, just on the new 13 370 theorems – more than half of which contain new terminology – the ratio of ENIGMA-proved to E-proved problems is 5934 to 3751, i.e., ENIGMA is here better than E by 58.20 %. These numbers show that the performance of our *anonymous* [25] logic-aware ML methods, which learn only from the structure of mathematical problems, is practically untouched by the transfer to the new setting with many new concepts and lemmas. This is quite unusual in today's machine learning which seems dominated by large language models that typically struggle on new terminology.

7 Proofs

As the main experiments progressed from spring 2020 to summer 2021, we have collected interesting examples of automatically found proofs and published their summary descriptions on our web page.[10] As of September 2021 there were over 200 of such example proofs, initially with ATP length in tens of clause steps, and gradually reaching hundreds of clause steps. Initially these were proofs found in the bushy setting, with proofs done in the chainy (premise-selection) setting added later, typically to show the effect of alternative premises.

One of the earliest proofs that we put on the web page ↗ is `NEWTON:72` ↗ proving that for every natural number there exists a larger prime:

```
for l being Nat ex p being Prime st p is prime & p > l
```

The ENIGMA proof ↗ starts from 328 preselected Mizar facts which translate to 549 initial clauses. The search is guided by a particular version of the GNN running at that time (April 2020) on the CPU. Since this is relatively costly, the proof search generated only 2856 nontrivial clauses in 6 s, doing 734 nontrivial given clause loops. The final proof takes 83 clausal steps, and uses 38 of the 328 initially provided steps. Many of them replay the

---

[10] https://github.com/ai4reason/ATP_Proofs

Table 5 The final Mizar hammer portfolio for 420 s.

t	base	ENIGMA	slice
2	bls0f17	GNN	$\mathcal{G}_{-1}$
2	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
2	bls0f17	GNN	$\mathcal{L}_{0.1}$
2	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
2	vampire	-	$\mathcal{N}_{uni}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
2	bls0f17	GNN	$\mathcal{N}_{eni}$
2	bls0f17	GNN	$\mathcal{N}_{eni}$
2	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
2	bls0f17	GNN	$\mathcal{G}_{-3}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	vampire	-	$\mathcal{K}^{au}_{var}$
2	mzr02	-	$\mathcal{K}^{cp}_{var}$
2	bls0f17	GNN	$\mathcal{G}_{0}$
2	vampire-16	-	$\mathcal{G}_{-5}$
2	vampire	-	$\mathcal{N}_{uni}$
2	bls0f17	GNN	$\mathcal{N}_{eni}$
5	bls0f17	GNN	$\mathcal{N}_{au}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
2	vampire-16	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
5	bls0f17	GNN	$\mathcal{L}_{0.05}$
2	vampire-18	-	$\mathcal{G}$
2	mzr22	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L}}_{.5,.2,.3}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,geo}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	vampire	-	$\mathcal{K}^{cp}_{var}$
2	BliStr-edc9	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L}}_{.5,.2,.3}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,chrono}$
2	vampire	-	$\mathcal{N}_{uni}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
10	mzr02	GNN	$\mathcal{L}_{0}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
5	bls0f17	GNN	$\mathcal{N}_{au}$
2	bls0f17	GNN	$\mathcal{N}_{eni}$
2	vampire-2	-	$\mathcal{G}_{16}$
2	vampire-16	-	$\mathcal{N}_{au}$
2	vampire	-	$\mathcal{N}_{au}$
2	E-auto	-	$\mathcal{L}_{96}$
10	bls05fc	GBDT	$\mathcal{L}_{0.25}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,min}$
2	vampire-16	-	$\mathcal{N}_{sub}$
2	vampire-16	-	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$

t	base	ENIGMA	slice
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
2	Blistr-5fce	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L}}_{.5,.2,.3}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	E-auto	-	$\mathcal{G}_{-2}$
2	vampire	-	$\mathcal{K}^{uni}_{var}$
2	vampire-16	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,geo}$
2	vampire	-	$\mathcal{K}^{uni}_{var}$
2	vampire-21	-	$\mathcal{G}$
2	vampire	-	$\mathcal{N}_{sub}$
2	vampire	-	$\mathcal{N}_{uni}$
2	E-auto	-	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
2	vampire	-	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,min}$
2	vampire	-	$\mathcal{N}_{cp}$
2	bls0f17	GNN	$\mathcal{K}^{eni}_{var}$
5	bls0f17	GNN	$\mathcal{L}_{0.01}$
10	bls05fc	GBDT	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.33,.33,.33}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
5	bls0f17	GNN	$\mathcal{N}_{eni}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
10	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
10	mzr03	GBDT	$\mathcal{G}_{64}$
10	bls05fc	GBDT	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.33,.33,.33}$
10	mzr02	GBDT	$\mathcal{K}_{short}$
10	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.5,.25,.25}$
10	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
10	bls0f17	GBDT	$\mathcal{L}_{0.01}$
5	bls0f17	GNN	$\mathcal{N}_{au}$
5	bls0f17	GNN	$\mathcal{L}_{0.005}$
10	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.5,.2,.2,.1}$
5	bls0f17	GNN	$\mathcal{L}_{0.01}$
5	bls0f17	GNN	$\mathcal{N}_{au}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
5	mzr03	-	$\mathcal{E}^{\mathcal{N},\mathcal{K},\mathcal{L},\mathcal{G}}_{.25,.25,.25,.25}$
10	bls0f17	GNN	$\mathcal{G}_{-1}$
5	bls0f17	GNN	$\mathcal{G}_{0}$
10	bls05fc	GBDT	$\mathcal{K}_{short}$
5	bls0f17	GNN	$\mathcal{N}_{au}$
5	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
10	bls0f17	GNN	$\mathcal{G}_{0.5}$
10	bls0f17	GNN	$\mathcal{N}_{eni}$
10	bls0f17	GBDT	$\mathcal{L}_{0.01}$
10	bls0f17	GBDT	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5,\&min}$
10	bls0f17	GNN	$\mathcal{N}_{au}$
10	bls0f17	GNN	$\mathcal{E}^{\mathcal{N},\mathcal{K}}_{.5,.5}$
10	bls0f17	GNN	$\mathcal{N}_{au}$
10	mzr03	GBDT	$\mathcal{E}^{\mathcal{N},\mathcal{G},\mathcal{K}}_{.33,.33,.33}$

arithmetical arguments done in Mizar. An interesting point is that the guided prover is here capable of synthesizing a nontrivial witness $(n! + 1)$ by using the supplied facts, after which the proof likely becomes reasonably straightforward given the knowledge in the library (see the Appendix for a more detailed discussion of this example). In general, using the supplied facts together with the trained learner for guided synthesis of nontrivial witnesses seems to be one of the main improvements brought by the ENIGMA guidance that contributed to the new proofs in comparison with the Mizar40 evaluation. This led us to start research of neural synthesis of witnesses and conjectures for AI/TP settings [13, 16, 17, 58].

Arithmetical reasoning, and other kinds of "routine computation" in general, have turned out to be areas where ENIGMA often gradually improved by solving increasingly hard Mizar problems and learning from them. Such problems include reasoning about trigonometric functions, integrals, derivatives, matrix manipulation, etc. From the more advanced results done by 3-phase ENIGMA, this is, e.g., a 619-long proof of `SINCOS10:86` ↗ found in 60 s, doing a lot of computation about the domain and range of arcsec ↗, and a 326-long proof of `FDIFF_8:14` ↗, found in 31 s, about the derivative of $\tan(\ln x)$ ↗.

```
for x being set st x in [.(- (sqrt 2)),(- 1).] holds arcsec2 . x in [.((3 / 4) * PI),PI.]
```

```
for Z being open Subset of REAL st Z c= dom (tan * ln) holds tan * ln
is_differentiable_on Z
& for x being Real st x in Z holds ((tan * ln) '| Z) . x = 1 / (x * (cos . (ln . x))^2)
```

The first proof uses 83 Mizar facts, starting with 1025 preselected ones. Its proof search took 5344 nontrivial given clauses and generated over 100k nontrivial clauses in total, making the 3-phase filtering and the use of the GPU server essential for finding the proof efficiently. The second proof uses 55 Mizar facts, 3136 given clause loops and it generated 26.6k nontrivial clauses. The reader can see on our web page that there are many solved problems of such "mostly computational" kind, suggesting that such learning approaches may be suitable for automatically gaining competence in routine computational tasks, without the need to manually program them as done, e.g., in SMT solvers. This has motivated our research in learning reasoning components [10]. Two less "computational" but still very long ATP proofs found by 3-phase ENIGMA are `BORSUK_5:31` ↗ saying that the closure of rationals on (a,b) is [a,b] ↗, and `IDEAL_1:22` ↗ saying that commutative rings are fields iff ideals are trivial ↗:

```
for A being Subset of R^1 for a, b being real number
st a < b & A = RAT (a,b) holds Cl A = [.a,b.]
```

```
for R being non degenerated comRing holds R is Field iff
for I being Ideal of R holds I = {(0. R)} or I = the carrier of R
```

The Mizar proof of `BORSUK_5:31` takes 80 lines. ENIGMA finds a proof from 38 Mizar facts that uses 359 clausal steps in 4883 given clause loops. On the 400k generated clauses, the multi-phase ENIGMA mechanisms work as follows. 133 869 clauses are frozen by parental guidance, 83 871 are then filtered by aggressive subsumption, and 64 364 by the first-stage LightGBM model. 125 489 remaining "good" clauses are gradually evaluated (in 176 batched calls) by the GNN server, using a context of 1536 processed clauses. The ENIGMA proof of `IDEAL_1:22` uses 48 Mizar facts and takes 493 clausal steps in 4481 given clause loops.

One example of an ATP proof made possible thanks to the premise selector noticing alternative lemmas in the library is `FIB_NUM2:69` ↗. This theorem, called in the MML "Carmichael's Theorem on Prime Divisors", states that if m divides the n-th Fibonacci number (Fib n), then m does not divide any smaller Fibonacci number, provided m, n are prime numbers ↗. The Mizar proof has 122 lines, uses induction and we cannot so far replay it with ATPs. The premise selector, however, finds a prior library lemma `FIB_NUM:5` ↗ saying that `(Fib m) gcd (Fib n) = Fib (m gcd n)`, from which the proof follows, using 159 clausal steps, 4214 given clause loops and 32 Mizar facts. Finally, an example of a long Deepire proof ↗ using a high time limit is `ORDINAL5:36` ↗, i.e., the $\epsilon_0 = \omega^{\omega^{\omega^{\cdots}}}$ formula for the zeroth epsilon ordinal ↗:

```
first_epsilon_greater_than 0 = omega |^|^ omega
```

The search took 38 065 given clause loops and 504 s. The proof has 1193 clausal steps, using 49 Mizar facts. Deepire's very efficient neural guidance took only 18 s of the total time here.

8   Conclusion: AI/TP Bet Completed

In 2014, after the 40 % numbers were obtained by Kaliszyk and Urban both on the Flyspeck and Mizar corpora, the last author publicly announced three AI/TP bets[11] in a talk at Institut Henri Poincare and offered to bet up to 10 000 EUR on them. Part of the second bet said that by 2024, 60 % of the MML and Flyspeck toplevel theorems will be provable automatically when using the same setting as in 2014. In the HOL setting, this was done as early as 2017/18 by the TacticToe system, which achieved 66.4 % on the HOL library in 60 s and 69 % in 120 s [14, 15]. One could however argue that TacticToe introduced a new kind of ML-guided tactical prover that considerably benefits from targeted, expert-written procedures tailored to the corpora. This in particular showed in the large boost on HOL problems that required induction, on which standard higher-order ATPs traditionally struggled.

In this work, we largely completed this part of the second AI/TP bet also for the Mizar library. The main caveat is our use of more modern hardware, in particular many ENIGMAs using the GPU server for clause evaluation. It is however clear (both from the LightGBM experiments and from the very efficient and CPU-based Deepire experiments) that this is not a major issue. While it is today typically easier to use dedicated hardware in ML-based experiments, there is also growing research in the extraction of faster predictors from those trained on GPUs that can run more efficiently on standard hardware.

References

1 Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

2 Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014. `doi:10.1007/s10817-013-9286-5`.

3 Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, and Karol Pak. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *J. Autom. Reason.*, 61(1-4):9–32, 2018. `doi:10.1007/s10817-017-9440-6`.

4 Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015. `doi:10.1007/978-3-319-20615-8_17`.

5 Grzegorz Bancerek and Piotr Rudnicki. A Compendium of Continuous Lattices in MIZAR. *J. Autom. Reasoning*, 29(3-4):189–224, 2002. `doi:10.1023/A:1021966832558`.

---

[11] `http://ai4reason.org/aichallenges.html`

6   Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning*, 57(3):219–244, 2016. `doi:10.1007/s10817-016-9362-8`.

7   Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016. `doi:10.6092/issn.1972-5787/4593`.

8   Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. `doi:10.1145/2939672.2939785`.

9   Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. `doi:10.3115/v1/d14-1179`.

10   Karel Chvalovský, Jan Jakubuv, Miroslav Olšák, and Josef Urban. Learning theorem proving components. In Anupam Das and Sara Negri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6-9, 2021, Proceedings*, volume 12842 of *Lecture Notes in Computer Science*, pages 266–278. Springer, 2021. `doi:10.1007/978-3-030-86059-2_16`.

11   Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. ENIGMA-NG: Efficient neural and gradient-boosted inference guidance for E. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2019. `doi:10.1007/978-3-030-29436-6_12`.

12   Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=SkZxCk-0Z`.

13   Thibault Gauthier. Deep reinforcement learning for synthesizing functions in higher-order logic. In *LPAR*, volume 73 of *EPiC Series in Computing*, pages 230–248. EasyChair, 2020.

14   Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC*, pages 125–143. EasyChair, 2017. `doi:10.29007/ntlb`.

15   Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, 2021. `doi:10.1007/s10817-020-09580-x`.

16   Thibault Gauthier, Miroslav Olšák, and Josef Urban. Alien coding. *CoRR*, abs/2301.11479, 2023.

17   Thibault Gauthier and Josef Urban. Learning program synthesis for integer sequences from scratch. *CoRR*, abs/2202.11908, 2022.

18   Zarathustra Goertzel, Jan Jakubův, and Josef Urban. ENIGMAWatch: ProofWatch meets ENIGMA. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 374–388, Cham, 2019. Springer International Publishing.

19   Zarathustra Amadeus Goertzel. Make E smart again (short paper). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 408–415. Springer, 2020.

20   Zarathustra Amadeus Goertzel, Karel Chvalovský, Jan Jakubuv, Miroslav Olšák, and Josef Urban. Fast and slow Enigmas and parental guidance. In *FroCoS*, volume 12941 of *Lecture Notes in Computer Science*, pages 173–191. Springer, 2021.

21 Zarathustra Amadeus Goertzel, Jan Jakubův, Cezary Kaliszyk, Miroslav Olšák, Jelle Piepenbrock, and Josef Urban. The Isabelle ENIGMA. In *ITP*, volume 237 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

22 Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

23 Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *J. Artificial Intelligence Research*, 36:267–306, October 2009.

24 Jan Jakubův and Josef Urban. Hierarchical invention of theorem proving strategies. *AI Commun.*, 31(3):237–250, 2018. `doi:10.3233/AIC-180761`.

25 Jan Jakubuv, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

26 Jan Jakubuv, Martin Suda, and Josef Urban. Automated invention of strategies and term orderings for vampire. In *GCAI*, volume 50 of *EPiC Series in Computing*, pages 121–133. EasyChair, 2017.

27 Jan Jakubuv and Josef Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017. `doi:10.1145/3018610.3018619`.

28 Jan Jakubuv and Josef Urban. Enhancing ENIGMA given clause guidance. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018. `doi:10.1007/978-3-319-96812-4_11`.

29 Jan Jakubuv and Josef Urban. Hammering Mizar by learning clause guidance. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ITP.2019.34`.

30 Cezary Kaliszyk and Josef Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In Jasmin Christian Blanchette and Josef Urban, editors, *PxTP 2013*, volume 14 of *EPiC Series*, pages 87–95. EasyChair, 2013.

31 Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014. `doi:10.1007/s10817-014-9303-3`.

32 Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015. `doi:10.1007/s10817-015-9330-8`.

33 Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, pages 3084–3090. AAAI Press, 2015.

34 Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Automating formalization by statistical and semantic parsing of mathematics. In *ITP*, volume 10499 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2017.

35 Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.

36 Artur Korniłowicz and Christoph Schwarzweller. Computers and algorithms in Mizar. *Mechanized Mathematics and Its Applications*, 4(1):43–50, 2005.

37 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013. `doi:10.1007/978-3-642-39799-8_1`.

38   Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: `https://openreview.net/forum?id=S1eZYeHFDS`.

39   Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020. `doi:10.3233/FAIA200244`.

40   David W. Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *J. Artif. Intell. Res.*, 11:169–198, 1999. `doi:10.1613/jair.614`.

41   Bartosz Piotrowski and Josef Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018. `doi:10.1007/978-3-319-94205-6_37`.

42   Bartosz Piotrowski and Josef Urban. Stateful premise selection by recurrent neural networks. In *LPAR*, volume 73 of *EPiC Series in Computing*, pages 409–422. EasyChair, 2020.

43   Bartosz Piotrowski, Josef Urban, Chad E. Brown, and Cezary Kaliszyk. Can neural networks learn symbolic rewriting? *CoRR*, abs/1911.04873, 2019.

44   John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001. URL: `https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning`.

45   Stephan Schulz. E – A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002. URL: `http://iospress.metapress.com/content/n908n94nmvk59v3c/`.

46   Stephan Schulz. Simple and efficient clause subsumption with feature vector indexing. In *Automated Reasoning and Mathematics*, volume 7788 of *Lecture Notes in Computer Science*, pages 45–67. Springer, 2013.

47   Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013. `doi:10.1007/978-3-642-45221-5_49`.

48   Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

49   Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Proc. of the 27th CADE, Natal, Brasil*, number 11716 in LNAI, pages 495–507. Springer, 2019.

50   Stephan Schulz and Geoff Sutcliffe. Proof generation for saturating first-order theorem provers. In David Delahaye and Bruno Woltzenlogel Paleo, editors, *All about Proofs, Proofs for All*, volume 55 of *Mathematical Logic and Foundations*, pages 45–61. College Publications, London, UK, January 2015.

51   Martin Suda. Improving ENIGMA-style clause selection while learning from history. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 543–561. Springer, 2021. `doi:10.1007/978-3-030-79876-5_31`.

52   Martin Suda. Vampire with a brain is a good ITP hammer. In Boris Konev and Giles Reger, editors, *Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings*, volume 12941 of *Lecture Notes in Computer Science*, pages 192–209. Springer, 2021. `doi:10.1007/978-3-030-86205-3_11`.

53   Tanel Tammet. Towards efficient subsumption. In *CADE*, volume 1421 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 1998.

54   J. Urban. Translating Mizar for First Order Theorem Provers. In A. Asperti, B. Buchberger, and J.H. Davenport, editors, *Proceedings of the 2nd International Conference on Mathematical Knowledge Management*, number 2594 in LNCS, pages 203–215. Springer, 2003.

55   Josef Urban. MPTP – Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004. `doi:10.1007/s10817-004-6245-1`.

56   Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006. `doi:10.1007/s10817-006-9032-3`.

57   Josef Urban. BliStr: The Blind Strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. Easy-Chair, 2015. URL: `http://www.easychair.org/publications/paper/BliStr_The_Blind_Strategymaker`, `doi:10.29007/8n7m`.

58   Josef Urban and Jan Jakubuv. First neural conjecturing datasets and experiments. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2020. `doi:10.1007/978-3-030-53518-6_24`.

59   Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 – Machine Learner for Automated Reasoning with Semantic Guidance. In *IJCAR*, pages 441–456, 2008. `doi:10.1007/978-3-540-71070-7_37`.

60   Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. `arXiv:1706.03762`.

61   Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *LNCS*, pages 255–270. Springer, 2018. `doi:10.1007/978-3-319-96812-4_22`.