BAIT: Benchmarking (Embedding) Architectures for Interactive Theorem-Proving

Sean Lamont[1, 2], **Michael Norrish**[1], **Amir Dezfouli**[3], **Christian Walder**[4], **Paul Montague**[2]
[1]Australian National University
[2]Defence Science and Technology Group
[3]BIMLOGIQ
[4]Google DeepMind
sean.lamont@anu.edu.au

Abstract

Artificial Intelligence for Theorem Proving has given rise to a plethora of benchmarks and methodologies, particularly in Interactive Theorem Proving (ITP). Research in the area is fragmented, with a diverse set of approaches being spread across several ITP systems. This presents a significant challenge to the comparison of methods, which are often complex and difficult to replicate.

Addressing this, we present BAIT, a framework for fair and streamlined comparison of learning approaches in ITP. We demonstrate BAIT's capabilities with an in-depth comparison, across several ITP benchmarks, of state-of-the-art architectures applied to the problem of formula embedding. We find that Structure Aware Transformers perform particularly well, improving on techniques associated with the original problem sets. BAIT also allows us to assess the end-to-end proving performance of systems built on interactive environments. This unified perspective reveals a novel end-to-end system that improves on prior work. We also provide a qualitative analysis, illustrating that improved performance is associated with more semantically-aware embeddings. By streamlining the implementation and comparison of Machine Learning algorithms in the ITP context, we anticipate BAIT will be a springboard for future research.

Introduction

Interactive Theorem Proving (ITP), a central paradigm of formal verification, has been used to write verified compilers (Leroy 2014; Tan et al. 2019), formalise mathematical conjectures (Gonthier 2008), and develop provably correct microkernels (Klein et al. 2009). As proficiency in both the formal system and application domain is needed, large verification projects require significant human resources and expertise. This restricts their scalability and widespread adoption, with, for example, the seL4 verification taking 25 person years (Klein et al. 2009). Advances in Artificial Intelligence for Interactive Theorem Proving (AI-ITP) have shown potential in automating and assisting human ITP guidance, but there remain many challenges in the area. With the current state of the art achieving 42% accuracy on the (highly difficult) miniF2F-curriculum benchmark, there is still much progress to be made (Lample et al. 2022; Zheng, Han, and Polu 2022).

To that end, it is important to efficiently and fairly compare approaches. As mentioned in (Yang et al. 2023), problems such as compute requirements and private code have made research in the area difficult. Adding to this difficulty is the fragmentation of results across ITP systems. Benchmarks and environments exist for HOL Light (Bansal et al. 2019a; Kaliszyk, Chollet, and Szegedy 2017), HOL4 (Wu et al. 2021a), Lean (Polu et al. 2023; Han et al. 2022; Yang et al. 2023), Isabelle (Li et al. 2021) and Metamath (Kaliszyk and Urban 2015). These provide a broad set of tasks for benchmarking. However being isolated to a single system complicates comparisons between them. This is magnified by the variety and complexity of the learning algorithms, which vary over several axes. For example, TACTICZERO (Wu et al. 2021a) uses a seq2seq autoencoder for expressions, and learns through online Reinforcement Learning (RL) with a custom goal selection algorithm. (Bansal et al. 2019a) instead use Breadth First Search (BFS) for goal selection, with offline learning over labelled proof logs.

Despite this, we show that many fundamental components are common across AI-ITP systems. We leverage these to develop BAIT, the first cross-platform, unified framework for AI-ITP research. BAIT brings together several environments, datasets and models in AI-ITP, with a central interface for experiments and sharing components between systems. Being fully open source, the addition of new benchmarks and algorithms is facilitated by a modular and decoupled design. BAIT combines automatic logging, checkpointing and configuration management to allow for rapid testing and prototyping of ideas with minimal overhead.

We use BAIT to study an important problem in AI-ITP, which is the choice of embedding architecture. The embedding model is critical, being used to encode ITP expressions for subsequent tactic, premise and goal selection. Current results either use graph based approaches (Kaliszyk, Chollet, and Szegedy 2017; Paliwal et al. 2020; Crouse et al. 2020), or treat expressions as a sequence (Lample et al. 2022; Polu et al. 2023; Han et al. 2022), with no thorough comparison between them across ITP systems. INT (Wu et al. 2021b) provides the only comparison, in a synthetic proving environment, without directly isolating the embedding architecture. We study approaches in various tasks across several ITPs, including recent Structure Aware Transformers (Chen, O'Bray, and Borgwardt 2022; Luo, Thost, and Shi 2023).

		Tactics		Learning				Representation	
Approach	ITP	Fixed	Gen	SL	RL	Pretrain	Search	Graph	Seq
Yang and Deng (2019)	Coq		✓	✓			BestFS	✓	
GAMEPAD (Huang et al. 2019)			✓	✓			BestFS	✓	
Bansal et al. (2019b)[1]	HOL Light	✓		✓			BFS	✓	
TACTICZERO (Wu et al. 2021a)	HOL4	✓			✓		Fringe		✓
TACTICTOE (Gauthier et al. 2021)		✓		✓			BestFS		✓
Wu et al. (2021b)	INT	✓	✓	✓			MCTS	✓	✓
Jiang et al. (2021)	Isabelle		✓	✓		✓	BestFS		✓
Polu et al. (2023)[2]	Lean		✓	✓		✓	BestFS		✓
REPROVER (Yang et al. 2023)		✓	✓	✓		✓	BestFS		✓
HTPS (Lample et al. 2022)			✓	✓		✓	MCTS*		✓
HOLOPHRASM (Whalen 2016)	Metamath	✓	✓	✓			UCT*		✓
Poesia and Goodman (2023)	Peano	✓			✓		BestFS*		✓

[1]Also Paliwal et al. (2020); Bansal et al. (2019a). [2]Also Polu and Sutskever (2020); Han et al. (2022). *: Modified

Table 1: An overview of current learning approaches in ITP. The ITP column indicates the underlying Proving System. The Tactics column indicates whether a generative model is used to generate the tactic. The Learning column indicates the use of Supervised Learning (SL), Reinforcement Learning (RL) and/or Pretrained Models. The Search column indicates the strategy for choosing which goal(s) to work on next. The Representation column indicates whether a graph or sequence representation is used. The variety of approaches and underlying systems apparent here motivates our unified platform.

To summarise, we present the following contributions:

- We introduce BAIT, the first cross-platform, open source framework for benchmarking results across ITP systems.
- We use BAIT to perform a comparison of embedding architectures in ITP, finding that Structure Aware Transformers (Chen, O'Bray, and Borgwardt 2022; Luo, Thost, and Shi 2023) improve upon Graph Neural Networks (GNNs) and Transformers across several datasets.
- We reveal a large improvement in the end-to-end TACTICZERO (Wu et al. 2021a) algorithm through experiments with the embedding architecture, with a qualitative analysis finding more semantically-aware embeddings.

Background

Related Work

The recent LEANDOJO (Yang et al. 2023) provides an open source ITP framework with easily reproducible results. As with other frameworks, such as HOList (Bansal et al. 2019a), INT (Wu et al. 2021b) and CoqGym (Yang and Deng 2019), it is focused on a single proving system. Our contribution is complementary, building upon the extensive work developing these systems to create a unified cross-platform framework. MWPToolkit and LILA (Lan et al. 2022; Mishra et al. 2022) unify approaches for Math Word Problems, a related area of AI for mathematical reasoning.

The prevailing embedding models in AI-ITP are GNNs and Transformers, with the only direct comparison in the synthetic INT (Wu et al. 2021b) framework. We extend this to multiple benchmarks, and study combinations of both architectures.

Benchmarks and Approaches in AI-ITP

Current benchmarks are based either on proxy tasks from proof logs, or on end-to-end proving performance with ITP interaction. A common proxy task is to predict lemmas useful for a goal, which is an important step in many ITP tactics. Such *premise selection* benchmarks include HOLStep (Kaliszyk, Chollet, and Szegedy 2017), MIZAR40 (Kaliszyk and Urban 2015) and ISARStep (Li et al. 2021). LeanStep (Han et al. 2022) includes premise selection and other tasks such as predicting masked subterms or types. HOList (Bansal et al. 2019a) contains premise and tactic selection data in HOL Light, based on both human and synthetic proof logs, with an end-to-end environment for evaluation. Other end-to-end benchmarks include CoqGym (Yang and Deng 2019), LeanGym (Polu et al. 2023), LeanDojo (Yang et al. 2023), LISA (Jiang et al. 2021) for Isabelle, and INT (Wu et al. 2021b), a synthetic proving system for AI-ITP research. miniF2F (Zheng, Han, and Polu 2022) contains Olympiad, undergraduate and high school mathematics problems in Lean, Metamath, HOL Light and Isabelle. The difficulty of problems has made this a key benchmark for performance in state-of-the-art systems (Lample et al. 2022; Yang et al. 2023; Polu et al. 2023).

From Table 1, we note the variety of learning approaches and their fragmentation across ITPs. For example, we see the exclusive use of fixed tactic models in HOL4 and HOL Light, while generative models are prevalent in Lean, Isabelle and Coq. This divide in approaches motivates a cross-platform framework, with the particularly large split in graph and sequence based representations further motivating a study of embedding architectures.

AI-ITP

The ITP learning task can be modelled as a sequential decision problem (Powell 2022; Wu et al. 2021a). The objective in each round is to prove an initial theorem, defined as the goal g. The initial state $(s_0 \supseteq g) \in \mathcal{S}$ includes g along with auxiliary information such as allowed premises and axioms. The model $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from the state to an action $a \in \mathcal{A}$. The environment $\mathcal{E} : \mathcal{A} \to \mathcal{S}$ extends the ITP system, taking an action a to a new state $s \in \mathcal{S}$, which is either a terminal state confirming the proof of the original goal, or represents a new state with subgoal(s) sufficient to prove g. We also assume that s includes all previous states such that the model has a full view of the history, making the problem a Markov Decision Process (MDP). The full state therefore represents all currently observed paths to proving the original goal, referred to as the *proof tree*. $\mathcal{E}$ can also include reward signals for reinforcement learning.

An AI-ITP system, such as the approaches in Table 1, defines how the model π is updated, given an interactive environment $\mathcal{E}$ and Data in the form of Proof Logs, as shown in Figure 1.

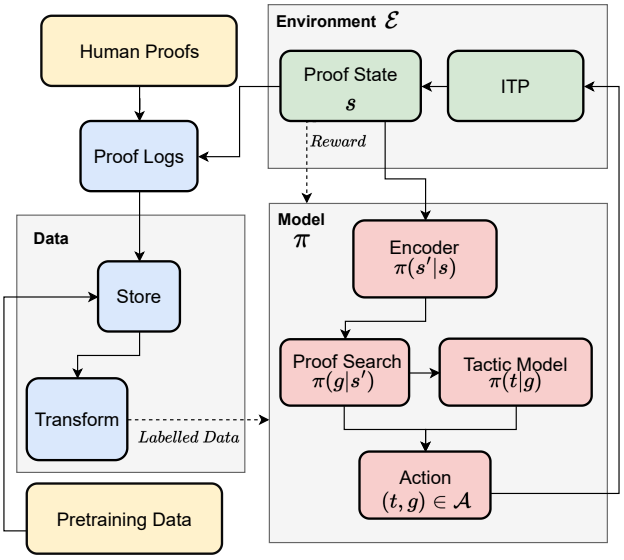

Figure 1: AI-ITP setup. A model π interacts with a proving environment $\mathcal{E}$, mapping a state s to an action (t, g), which defines tactic(s) t to apply to goal(s) $g \subseteq s$. π is trained with rewards or Data from processed Proof Logs, sourced from human proofs or agent-environment interactions.

Learning Approach The most common learning approach is supervised learning over proof logs, collected either from human proofs or from interaction of the AI system with the ITP environment. From these logs, labelled objectives such as premise and tactic selection are generated for training. Combining both approaches has been shown to be superior in several cases (Lample et al. 2022; Bansal et al. 2019b; Polu et al. 2023).

Reinforcement Learning (RL) has also been used, with the agent learning from a reward signal generated from interaction with the environment. Far less common than supervised approaches in ITP, RL has the advantage of removing biases in the labelling of data. For example, it is common to assign negative labels to premises not used in an original human proof, despite the potential for these to be useful in a different proof of the same conjecture (Kaliszyk, Chollet, and Szegedy 2017; Kaliszyk and Urban 2015; Bansal et al. 2019a). However, RL can suffer from high variance in the gradient updates (Sutton and Barto 2018), and requires appropriately shaping the reward to optimise proof performance. It remains unclear which is superior in AI-ITP, further motivating a centralised comparison platform.

Pretrained language models have also been used, which are generally finetuned on proof data as in (Lample et al. 2022; Polu et al. 2023; Han et al. 2022).

Encoder The encoder model $\pi(s'|s) : \mathcal{S} \to \mathbb{R}^D$ maps the current proof state s into Euclidean space, with $D \in \mathbb{N}$ depending upon the approach. An effective embedding model is critical, as it is the only information used by the tactic and goal selection models. Approaches either treat the current goals $g \subseteq s$ as a Natural Language sequence (Lample et al. 2022; Polu et al. 2023; Wu et al. 2021a), or use a graph based representation (Bansal et al. 2019b; Paliwal et al. 2020; Wang et al. 2017; Evans et al. 2018). The resulting embedding is $s' \in \mathbb{R}^{n \times k \times d}$ where n, k, d are the number of goals, tokens and embedding dimensions respectively. The tokens are often pooled so as to produce a single vector for each goal, such that $s' \in \mathbb{R}^{n \times d}$.

Proof Search Proof search follows state embedding, as shown in Figure 1. It is the sub-model $\pi(g|s')$ mapping from the encoded state $s' \in \mathbb{R}^D$ to a subset of goals $g \subseteq s'$. A majority of approaches apply Breadth First Search (Bansal et al. 2019a; Huang et al. 2019) or Best First Search for goal selection. TACTICZERO (Wu et al. 2021a) showed an improvement by considering the likelihood of proving distinct sets of goals (fringes) equivalent to the original goal. Improvements have also been found through variations of Monte Carlo Tree Search algorithms (Lample et al. 2022; Wu et al. 2021b).

Tactic Model Tactic selection maps the selected goals to an action, with $\pi(t|g) : \mathbb{R}^D \to \mathcal{T}$. ITP tactics are usually a small, fixed set, however they may include arguments which can be arbitrary expressions. It is therefore common to restrict arguments to only use other expressions, lemmas or terms available in the current context. Selecting these arguments is a particularly important problem, known as *premise selection*. For these cases, the tactic model consists of predicting the tactic to apply, followed by the arguments conditioned on the tactic. Such approaches are marked as 'Fixed' in Table 1. Despite a much larger output space, generative models such as Transformer Decoders have also been applied to predict the entire tactic and argument as a text sequence, listed under 'Gen' in Table 1. Holophrasm (Whalen 2016) and ReProver (Yang et al. 2023) combine these approaches, with premise arguments restricted to a fixed set and the remaining tactic tokens being generated.

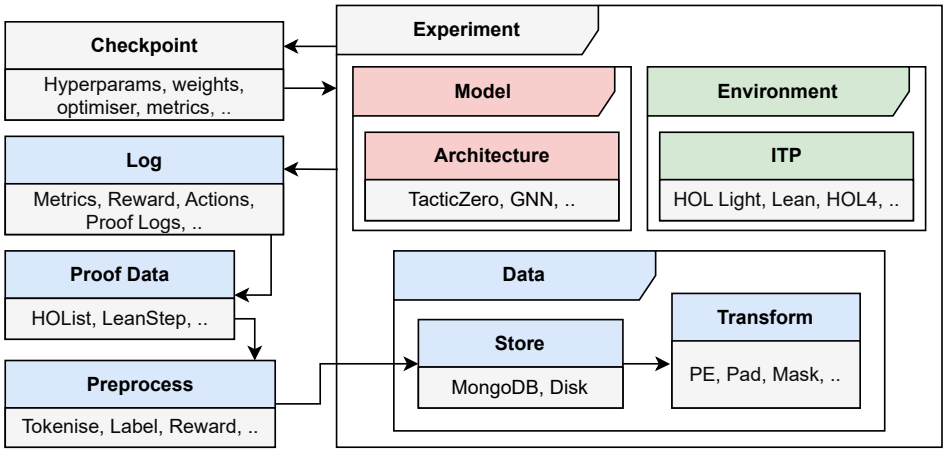

Figure 2: Overview of BAIT. The Experiment module abstractly defines tasks such as premise selection or RL. Experiment instances take a configuration specifying the Data, Model and Environment instances to use, with Checkpoints and Logs as output. Proof Data, either from existing sources or generated logs, is processed, stored and transformed for input to the Model.

BAIT

BAIT[1] is designed to be a general framework for AI-ITP, with Data, Model and Environment modules implementing the setup in Figure 1. These are managed with an additional Experiment module, as outlined in Figure 2.

Data Data modules process, store and transform proof data. Raw data is sourced from human proofs and current datasets, or through proof logs generated by experiment runs. It is then pre-processed and stored. Pre-processing here includes, for example, generating labelled training splits, tokenisation and converting expressions into a standardised graph (or sequence) format. Final batch transformations, such as padding and masking for sequence models, are then applied by a data loader during an Experiment.

Environment The Environment module embodies and extends the underlying ITP system. Support here includes converting model actions to the correct ITP format, processing the ITP state to be parsed by the model, and generating rewards for RL agents.

Model This module implements architectures for the Encoder, Proof Search and Tactic Models in Figure 1. Models are exposed to Experiments with a simple interface, which is straightforward to extend for integrating new architectures.

Experiment Experiments in BAIT define tasks in AI-ITP. They take as input a Model, Data source and possibly Environment to interact with. We currently include Experiment classes for premise selection across several benchmarks, as well as the HOList (Paliwal et al. 2020) training and evaluation setup, TACTICZERO RL setup (Wu et al. 2021a) and the INT (Wu et al. 2021b) experiments.

Experiments inherit a single, shared implementation of logging and checkpointing to streamline their development. Logging is done using Weights and Biases (Biewald 2020) which records all hyperparameters and metrics, with automatic plots, dashboards and run comparisons to ensure transparent and replicable results.

Once implemented, Experiments are manged using Hydra (Yadan 2019), which allows for complex configurations without modifying code. Hyperparameter sweeping is also simplified using these configurations.

We provide more details and example usage of BAIT in the supplementary material.

[1] https://github.com/sean-lamont/bait

Embedding Architectures

To demonstrate its utility as a research platform, we use BAIT to study embedding architectures in ITP. As we have noted, referring to Table 1, there is no clear comparison of embedding approaches for different representations in ITP. We address this by comparing state-of-the-art approaches, for both graph and sequence representations, in several supervised and end-to-end ITP benchmarks.

Representations and Approaches

Embedding models in ITP are either graph or sequence based, depending on the underlying representation.

Sequences Sequences in ITP are either human readable (pretty-printed), or structured s-expressions, which guarantee non-ambiguous operator precedence.

The current state-of-the-art model for sequences in ITP is the Transformer (Vaswani et al. 2017). The full Encoder-Decoder architecture has been used in end-to-end proving environments by (Lample et al. 2022; Wu et al. 2021b). We also note that GPT style Decoder only architectures have been used in (Polu and Sutskever 2020; Polu et al. 2023; Han et al. 2022), who predict all tactic tokens directly. As noted by (Yang et al. 2023), this is a bottleneck for premise selection and limits model generalisation, which was also observed in (Wu et al. 2021b). Hence the Transformer Encoder remains an important part of the architecture, and is the model we use for our sequence experiments.

Task	SAT	Directed SAT	GNN	Transformer	Ensemble	Bag of Words
HOList Tactic	35.2	39.1	35.5	38.2	**39.5**	32.1
HOList Relative Parameter	98.0	98.2	98.3	98.3	**98.7**	98.0
HOList top-5	81.2	**85.1**	81.8	83.5	84.8	78.7
MIZAR40 Premise Selection	73.6	**74.1**	73.5	73.7	73.5	72.3
HOLStep Premise Selection	**90.9**	*	90.3	89.5	90.6	75.7
LeanStep Premise Selection	96.6	**97.0**	95.7	95.9	96.1	92.1
HOL4 Premise Selection	91.0	91.1	91.6	**91.8**	91.8	87.4

Table 2: Accuracy benchmarks for embedding architectures across Supervised Learning Tasks. *Computationally intractable

Graphs For graph representations, expressions are parsed into abstract syntax trees, followed by various transformations to create a directed acyclic graph, such as variable renaming and subexpression sharing. See (Paliwal et al. 2020) or (Wang et al. 2017) for an explanation and comparison of different graph transformations, with additional details and examples in the supplementary material.

GNNs are the current state-of-the-art for graph based forumula embeddings, improving upon LSTM, CNN and WaveNet based approaches across several benchmarks (Paliwal et al. 2020; Wang et al. 2017; Bansal et al. 2019b; Crouse et al. 2020). The specific architectures used in ITP are generally variations on Message Passing Neural Networks (MPNN), adapted specifically for ITP formulae graphs (Paliwal et al. 2020; Wang et al. 2017; Bansal et al. 2019b). We briefly summarise the basic concept here:
Graph representations of expressions are given by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ representing variables, constants and functions, and edges $\mathcal{E}$ mapping from functions to their arguments. The MPNN is parameterised by T layers, or hops, with each layer aggregating *messages* from the immediate neighbors in the graph. Messages are treated differently from node parents $\mathcal{N}^-$ and children $\mathcal{N}^+$, so as to model the directed nature of an expression graph. Edges $e_{ij} \in \mathbb{N}$ from node i to node j represent the order of arguments, and are initially projected into $\mathbb{R}^{\mathbb{N}}$ by a learnable embedding. At each step $t \in [0, T]$, the embedding x_i of a node $i \in \mathcal{V}$ is

$$x_i^t = F_O\left(x_i^{t-1}, \mathcal{M}^t\right)$$

With message $\mathcal{M}^t$ given by

$$F_A\left(x_i^{t-1}, \sum_{j \in \mathcal{N}^-(i)} F_P(x_j^{t-1}, e_{ji}), \sum_{j \in \mathcal{N}^+(i)} F_C(x_j^{t-1}, e_{ij})\right)$$

F_A, F_P, F_C, F_O are learnable functions, usually taken to be Multi Layer Perceptrons (MLPs), and the final node embeddings are the output of the final layer $\{x_i^T\}_{i \in \mathcal{V}}$.

Structure Aware Transformer (*SAT*) models (Chen, O'Bray, and Borgwardt 2022) use the same graph representation as a GNN. The graph is first processed by an arbitrary GNN model, followed by a Transformer Encoder layer over the output. This is repeated for $T \in \mathbb{N}$ layers to produce the embeddings. *Directed SAT* modifies this to restrict attention for each node to only include ancestors or descendent nodes, which has been found to improve performance for directed graph problems (Luo, Thost, and Shi 2023).

Experiments

Representations For sequences, formulae are represented in s-expression format, which includes type information.

For graphs, we take the s-expression sequences and parse them into abstract syntax trees. Graphs are then processed with subexpression sharing, as outlined in (Paliwal et al. 2020). Using the same s-expression format in both sets of experiments ensures precedence, type and token information is kept constant between representations.

Model Details Each of the $n \in \mathbb{N}$ tokens in an expression is initially represented with a one-hot vector, followed by an Embedding layer projecting these to the dimension $d \in \mathbb{N}$ of the network. The initial representations $\{x_i^0\}_{i=1}^n$ are then used as input for the embedding model. The final token embeddings $\{x\}_{i=1}^n$ are aggregated through sum, max or mean pooling to produce a single embedding vector $e \in \mathbb{R}^d$.

Our Transformer Encoder follows the approach in (Vaswani et al. 2017), with sinusoidal positional encodings. We set a maximum sequence length of 1024.

For GNNs, we use variations of the MPNN architecture in the previous section, as well as Graph Convolutional Networks (GCN) (Zhang et al. 2019). We also include results from a 0-layer GNN, as a simple Bag of Words baseline.

Our SAT models use the implementations in (Chen, O'Bray, and Borgwardt 2022; Luo, Thost, and Shi 2023), with no positional encoding. For Directed SAT, computing the ancestor/descendent nodes is required preprocessing, which can be expensive for large datasets. We were therefore unable to test this approach on the HOLStep benchmark.

We also study an Ensemble which combines the best performing GNN and Transformer in each case. We implement this with an additional 2 layer MLP in the network, which concatenates the GNN and Transformer embeddings and projects them back into the original embedding dimension.

Supervised Setting

The benchmarks we use for supervised experiments are HOLStep (Kaliszyk, Chollet, and Szegedy 2017), MIZAR40 (Kaliszyk and Urban 2015), LeanStep (Han et al. 2022) and HOList (Bansal et al. 2019a). We also include a new HOL4 premise selection task, which we generate using theorem dependencies from the HOL4 standard library.

For premise selection experiments, embeddings are concatenated and fed through a 2-layer MLP with a final sigmoid layer and binary cross entropy as the loss.

We perform a hyperparameter sweep over configurations for all architectures. The best scoring checkpoint on the validation set was then assessed on the test set.

For HOList (Bansal et al. 2019a), we follow the exact supervised training setup described in (Paliwal et al. 2020). A training point consists of a goal, tactic, positive premise and a list of negative premises, constructed by randomly sampling from the set of all used premises. The tasks are to predict the ground truth tactic and premise. We keep their loss function, which is a weighted sum over these objectives. The architecture, including the GNN, is identical to what is presented in (Paliwal et al. 2020). Table 2 reports the tactic accuracy (Tactic), the number of times a positive premise is ranked higher than a negative (Relative Parameter), and whether the true tactic is in the top-5 predicted by the model (top-5) for the validation set.

As in previous work, we use accuracy as the performance metric (Paliwal et al. 2020; Wang et al. 2017). The full set of hyperparameters is in the supplementary material, with further details on the datasets and model architectures.

Results From Table 2, we see that the SAT models perform strongly in most benchmarks. The directed SAT model outperformed base SAT in all tested cases, highlighting the importance of modelling the directed structure. The HOL4 task is an interesting exception. We hypothesise the small size of the dataset, with only 7000 unique expressions, is likely insufficient for the complex SAT model to learn beyond simpler approaches. Transformers outperformed GNN only approaches overall, and Ensembles almost always outperformed the base GNN and Transformer models on which they were based. The Bag of Words model performed poorly on all benchmarks with the exception of the HOList relative parameter accuracy. (Paliwal et al. 2020) also found good results using a Bag of Words model in HOList, when implemented as a 0 layer GNN.

Despite the strong results, there are additional considerations to using SAT models in a practical AI-ITP system. It is far less efficient than current Transformer implementations, with a GNN pass in every layer complicating the optimisation of the attention computation. Directed SAT also requires computing ancestor and child nodes. Although this can be pre-computed, the vast majority of expressions generated in online agent-environment interaction are previously unseen.

Our results are consistent with previous work, validating our implementation of BAIT. For HOLStep, the GNN we base on (Wang et al. 2017) performs similarly to their reported results. For HOList, our results are consistent with the numbers reported in (Bansal et al. 2019a), with 'around 1%' error in relative parameter accuracy (where we have 1.3%–2%), and a 39–41% tactic accuracy. Their results also include synthetic proof logs to improve performance, while we only trained with human proof logs. The current state-of-the-art for the HOLStep and MIZAR40 benchmarks remains the DAG-LSTM from (Crouse et al. 2020). As this architecture cannot compute separate embeddings for goals and premises, which makes prediction computationally intractable for end-to-end systems (Wu et al. 2021a; Paliwal et al. 2020; Yang et al. 2023), we omit this approach.

Model	Cumulative	Valid (pass@1)
GNN	96.2%	**64.6%**
Transformer	**96.8%**	63.3%
Original TACTICZERO	90.7%	43.0%

Table 3: Goals proven by TACTICZERO in HOL4, after 1 attempt for validation and cumulatively for training.

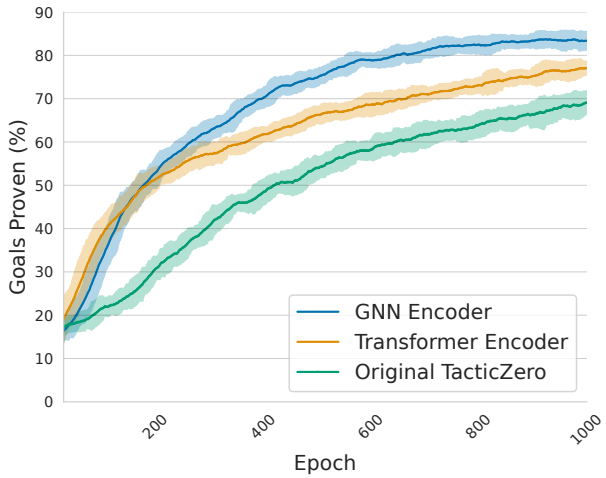

Figure 3: Goals proven during training by TACTICZERO, with different underlying embedding architectures.

End-to-End Setting

We use TACTICZERO (Wu et al. 2021a) to study embedding architectures in an end-to-end setting. TACTICZERO learns using only RL and interaction with the ITP environment. It is therefore a good benchmark for investigating embedding architectures beyond supervised problems. The original model uses a seq2seq autoencoder, trained on the reconstruction loss of the expressions. As the full training loop takes approximately 3 weeks, we test only a GNN and Transformer based Encoder. We pretrain these models on the HOL4 premise selection task, ensuring goals used for this problem were excluded in the data generation. We take 1185 goals from the original paper, which are theorems from the HOL4 standard library. We follow their protocol of a random 80:20 training and validation split, with 948 training goals and 237 validation goals. Figure 3 plots the number of goals proven from the training set per epoch, and Table 3 tabulates the cumulative (proven at least once) and validation (pass@1) results, which are the standard metrics for end-to-end tasks (Bansal et al. 2019b).

Results From Figure 3 and Table 3, we observe a large improvement in TACTICZERO when GNN or Transformer encoders are used, as compared to the original approach. The difference in validation performance is especially notable, with an approximate 50% increase in goals proven by the GNN model compared to the original seq2seq autoencoder. This suggests that the autoencoder model is overfitting to the training set and struggling to generalise to unseen goals.

Expression	GNN Encoder (Our Approach)	Original TacticZero
$\mathsf{diag}(A) = \mathsf{diag}(A^T)$	$R = (R^T)^T$	$\mathsf{FINITE}(\mathsf{POW}(s)) \Leftrightarrow \mathsf{FINITE}(s)$
$R\ x\ y \Rightarrow \mathsf{RC}(R)\ x\ y$	$\mathsf{RC}(\mathsf{RC}(R)) = \mathsf{RC}(R)$	$R\ x\ y \Rightarrow \mathsf{EQC}(R)\ x\ y$
$s \subseteq t \Leftrightarrow s \cup t = t$	$s\ \mathsf{DIFF}\ t = \emptyset \Leftrightarrow s \subseteq t$	$\mathsf{SURJ}\ f\ s\ t \Leftrightarrow \mathsf{IMAGE}\ f\ s = t$
$s \cup t = t \cup s$	$s \cup (t \cup u) = (s \cup t) \cup u$	$s \cap t = t \cap s$
$(s \cup t)x \Leftrightarrow x \in s \vee x \in t$	$x \in s \cup t \ \Leftrightarrow\ x \in s \vee x \in t$	$(s \cap t)x \Leftrightarrow x \in s \wedge x \in t$

Table 4: A selection of mathematical expressions (left) along with the nearest expression by cosine distance according to the TacticZero embedding (right) and GNN embedding (center).

As with our supervised experiments, when comparing the Transformer and GNN we find only a small difference in performance. INT (Wu et al. 2021b), found that Encoder-Decoder Transformers do not generalise as well as GNNs, which they suggest is due to the Decoder. For our case, we are only using a Transformer Encoder, with the rest of the architecture fixed. Our results are consistent with their hypothesis, with little difference between the GNN and Transformer (when excluding the Decoder) when generalising to unseen proofs in the Validation set.

The embedding model used by the original TacticZero is a fixed autoencoder trained only on the reconstruction loss of the original expression. Hence it is unsurprising to see an improvement in the end-to-end performance when compared to the other approaches. The original TacticZero result reports 49.2% of goals proven, compared to our observed 43%. Since we train for 200 additional epochs, it is possible that these additional epochs cause the model to overfit further. Regardless, our best approach is still a 31% improvement over their result, which highlights the critical role of the embedding model in the overall system.

Qualitative Analysis

With the aim of explaining the large observed improvement in TacticZero, we investigate the embeddings produced by the different models. We take a random set of expressions, generating their corresponding embeddings with the original TacticZero autoencoder and the best performing GNN Encoder from Figure 3. We then find the nearest expression as judged by cosine distance in each of the embedding spaces. In a majority of cases, the nearest neighbor as judged by the GNN model was far more semantically relevant to the original expression than the nearest neighbor as judged by the original TacticZero Autoencoder. Table 4 shows a small, insightful set of such cases, with a larger list in the Appendix (with full type information). We observe in several cases that the original TacticZero encoder finds expressions which are not semantically similar at all, such as the third row in Table 4 where TacticZero's encoder mentions constants (SURJ and IMAGE) that do not occur in the original expression, and fails to mention either subset or union, which do. In contrast, the GNN encoder finds an expression relating directly to subsets, and in this case it is easy to see how that expression might be useful in the proof of the original. We encourage the reader to compare other examples here and in the supplementary material. This provides some insight to the poor generalisation ability of the original encoder. If the nearest points in embedding space are semantically unrelated, then it is more likely for an unseen goal to be conflated with an unrelated expression.

Limitations

Due to computational constraints, we were unable to extensively evaluate HOList models on the end-to-end prover, with previous approaches using, for example, 2000 CPUs (Bansal et al. 2019b). We nevertheless include a small scale evaluation in the supplementary material as a sanity check on our implementation.

We did not evaluate several previous embedding approaches such as Tree-LSTMs or CNNs as there are several results indicating they are inferior to comparable GNN and Transformers (Paliwal et al. 2020; Wu et al. 2021b).

Conclusion

We introduce Bait, the first cross-platform framework for experimentation in Artificial Intelligence for Interactive Theorem Proving. Using Bait, we compared modern embedding architectures over several supervised benchmarks, finding that Structure Aware Transformers and Ensemble approaches outperform GNN and Transformer baselines. Extending our analysis to end-to-end systems, we found a large improvement over previous work and observed more semantically accurate embeddings were produced as a result.

Future Work

For better coverage across different ITPs and learning algorithms, Bait needs yet more benchmarks and datasets. LeanDojo (Yang et al. 2023) is a good candidate, being open source with state-of-the-art results on MiniF2F (Zheng, Han, and Polu 2022). Bait was designed to compare all components of AI-ITP, so comparing the learning, tactic selection and proof search approaches in a similar manner is a natural next step. Given the growth in research of pretrained LLMs for ITP, integrating these is another promising direction. Bait could also be used to investigate transfer tasks between systems, such as (Gauthier and Kaliszyk 2015).

Acknowledgements

We would like to acknowledge Defence Science and Technology Group (DSTG) for their support in this project. We would also like to thank Minchao Wu for his help with the TACTICZERO source code.

References

Bansal, K.; Loos, S.; Rabe, M.; Szegedy, C.; and Wilcox, S. 2019a. Holist: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, 454–463. PMLR.

Bansal, K.; Szegedy, C.; Rabe, M. N.; Loos, S. M.; and Toman, V. 2019b. Learning to Reason in Large Theories without Imitation. ArXiv:1905.10501.

Biewald, L. 2020. Experiment Tracking with Weights and Biases. Software available from wandb.com.

Chen, D.; O'Bray, L.; and Borgwardt, K. M. 2022. Structure-Aware Transformer for Graph Representation Learning. In *International Conference on Machine Learning*.

Crouse, M.; Abdelaziz, I.; Cornelio, C.; Thost, V.; Wu, L.; Forbus, K.; and Fokoue, A. 2020. Improving Graph Neural Network Representations of Logical Formulae with Subgraph Pooling. ArXiv:1911.06904.

Evans, R.; Saxton, D.; Amos, D.; Kohli, P.; and Grefenstette, E. 2018. Can Neural Networks Understand Logical Entailment? In *International Conference on Learning Representations*.

Gauthier, T.; and Kaliszyk, C. 2015. Sharing HOL4 and HOL Light Proof Knowledge. In *Logic Programming and Automated Reasoning*.

Gauthier, T.; Kaliszyk, C.; Urban, J.; Kumar, R.; and Norrish, M. 2021. TacticToe: Learning to Prove with Tactics. *Journal of Automated Reasoning*, 65(2): 257–286.

Gonthier, G. 2008. The Four Colour Theorem: Engineering of a Formal Proof. In Kapur, D., ed., *Computer Mathematics*, Lecture Notes in Computer Science, 333–333. Berlin, Heidelberg: Springer. ISBN 978-3-540-87827-8.

Han, J. M.; Rute, J.; Wu, Y.; Ayers, E.; and Polu, S. 2022. Proof Artifact Co-Training for Theorem Proving with Language Models. In *International Conference on Learning Representations*.

Huang, D.; Dhariwal, P.; Song, D.; and Sutskever, I. 2019. GamePad: A Learning Environment for Theorem Proving. In *International Conference on Learning Representations*.

Jiang, A. Q.; Li, W.; Han, J. M.; and Wu, Y. 2021. LISA: Language models of ISAbelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*.

Kaliszyk, C.; Chollet, F.; and Szegedy, C. 2017. HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving. In *International Conference on Learning Representations*.

Kaliszyk, C.; and Urban, J. 2015. MizAR 40 for Mizar 40. *Journal of Automated Reasoning*, 55(3): 245–256.

Klein, G.; Elphinstone, K.; Heiser, G.; Andronick, J.; Cock, D.; Derrin, P.; Elkaduwe, D.; Engelhardt, K.; Kolanski, R.; Norrish, M.; Sewell, T.; Tuch, H.; and Winwood, S. 2009. seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, 207–220. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-60558-752-3.

Lample, G.; Lacroix, T.; anne Lachaux, M.; Rodriguez, A.; Hayat, A.; Lavril, T.; Ebner, G.; and Martinet, X. 2022. HyperTree Proof Search for Neural Theorem Proving. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *Advances in Neural Information Processing Systems*.

Lan, Y.; Wang, L.; Zhang, Q.; Lan, Y.; Dai, B. T.; Wang, Y.; Zhang, D.; and Lim, E.-P. 2022. MWPToolkit: An Open-Source Framework for Deep Learning-Based Math Word Problem Solvers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11): 13188–13190. Number: 11.

Leroy, X. 2014. The CompCert C verified compiler: Documentation and user's manual. Avalailable at https://compcert.org/man/manual.pdf.

Li, W.; Yu, L.; Wu, Y.; and Paulson, L. C. 2021. IsarStep: a Benchmark for High-level Mathematical Reasoning. In *International Conference on Learning Representations*.

Luo, Y.; Thost, V.; and Shi, L. 2023. Transformers over Directed Acyclic Graphs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Mishra, S.; Finlayson, M.; Lu, P.; Tang, L.; Welleck, S.; Baral, C.; Rajpurohit, T.; Tafjord, O.; Sabharwal, A.; Clark, P.; and Kalyan, A. 2022. LILA: A Unified Benchmark for Mathematical Reasoning. In Goldberg, Y.; Kozareva, Z.; and Zhang, Y., eds., *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 5807–5832. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics.

Paliwal, A.; Loos, S.; Rabe, M.; Bansal, K.; and Szegedy, C. 2020. Graph Representations for Higher-Order Logic and Theorem Proving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03): 2967–2974. Number: 03.

Poesia, G.; and Goodman, N. D. 2023. Peano: learning formal mathematical reasoning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2251): 20220044.

Polu, S.; Han, J. M.; Zheng, K.; Baksys, M.; Babuschkin, I.; and Sutskever, I. 2023. Formal Mathematics Statement Curriculum Learning. In *The Eleventh International Conference on Learning Representations*.

Polu, S.; and Sutskever, I. 2020. Generative Language Modeling for Automated Theorem Proving. ArXiv:2009.03393.

Powell, W. B. 2022. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. John Wiley & Sons. ISBN 978-1-119-81505-1. Google-Books-ID: 6ahsEAAAQBAJ.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN 978-0-262-03924-6.

Tan, Y. K.; Myreen, M. O.; Kumar, R.; Fox, A.; Owens, S.; and Norrish, M. 2019. The verified CakeML compiler backend. *Journal of Functional Programming*, 29: e2. Publisher: Cambridge University Press.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wang, M.; Tang, Y.; Wang, J.; and Deng, J. 2017. Premise Selection for Theorem Proving by Deep Graph Embedding. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Whalen, D. 2016. Holophrasm: a neural Automated Theorem Prover for higher-order logic. ArXiv:1608.02644.

Wu, M.; Norrish, M.; Walder, C.; and Dezfouli, A. 2021a. TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning. In *NeurIPS*, volume 34.

Wu, Y.; Jiang, A. Q.; Ba, J.; and Grosse, R. 2021b. INT: An Inequality Benchmark for Evaluating Generalization in Theorem Proving. In *ICLR*.

Yadan, O. 2019. Hydra - A framework for elegantly configuring complex applications. Software available at https://github.com/facebookresearch/hydra.

Yang, K.; and Deng, J. 2019. Learning to prove theorems via interacting with proof assistants. In *36th International Conference on Machine Learning, ICML 2019*, 36th International Conference on Machine Learning, ICML 2019, 12079–12094. International Machine Learning Society (IMLS). 36th International Conference on Machine Learning, ICML 2019 ; Conference date: 09-06-2019 Through 15-06-2019.

Yang, K.; Swope, A.; Gu, A.; Chalamala, R.; Song, P.; Yu, S.; Godil, S.; Prenger, R.; and Anandkumar, A. 2023. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Neural Information Processing Systems (NeurIPS)*.

Zhang, S.; Tong, H.; Xu, J.; and Maciejewski, R. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1): 11.

Zheng, K.; Han, J. M.; and Polu, S. 2022. miniF2F: a cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations*.