Machine Learning Guided Equality Saturation

Nicole Heinimann
heinimann@tu-berlin.de
Technische Universität Berlin
Germany

Thomas Kœhler
thomas.koehler@cnrs.fr
ICube Lab, CNRS, Université de Strasbourg
France

Michel Steuwer
michel.steuwer@tu-berlin.de
Technische Universität Berlin
Germany

Abstract

Equality saturation has successfully been applied in many domains. Yet, scaling issues hold back its success in even more applications. The underlying e-graph data structure can grow rapidly quickly consuming all available resources.

Guided Equality Saturation [4] proposed a solution by breaking challenging rewrite problems into a sequence of equality saturations. This enables the technique to scale further and solve complex rewrite problems far out of reach of standard equality saturation. However, this technique relies on the human experts to provide insights in the form of *guides* that describe when to stop one equality saturation and start the next.

In this talk, we are going to present our ongoing efforts to reduce the reliance on human experts. In our *Machine Learning Guided Equality Saturation*, the ambition is to automatically generate guides using a machine learning model to enable the scaling of Equality Saturation to more complex applications. We report on the current state of our research and the machine learning model we are developing.

1 Introduction

E-Graphs are a clever data structure powering *Equality Saturation*, an equational rewrite approach, that forms the basis of optimizing compilers and automated proof searches. Equality saturation overcomes downsides of naive greedy rewrite systems. Still, e-graphs face scaling issues when dealing with real-world problems where just a single application of a rule can easily double the size of the e-graph, leading to a runaway scenario that can quickly consume all available memory. One source of the scaling issues are *explosive rewrites* that interact in unfortunate ways resulting in rapid growth of the e-graph. Simply removing those rewrites rules is not possible without losing the ability to prove a large number of equalities or massively restricting optimizations.

Multiple strategies have been proposed to mitigate the scaling issue. One possible approach consists of replacing more general explosive rewrites with more domain-specific rules that only apply to a subset of terms. Another approach uses a careful rule scheduler that applies explosive rewrite rules as sparingly as possible. Changing the e-graph itself to be more efficient and resilient against blow-up in certain settings has also been explored [5]. However, while these techniques improve efficiency, none provides a principled solution to address the scaling issue.

2 Expert Guided Equality Saturation

To address the scalability issue without restricting expressivity or requiring more resources, another approach is *Guided Equality Saturation* [4] which allows users to supply *guides* $G_1 \dots G_n$. In contrast to standard equality saturation, *Guided Equality Saturation* does not attempt to reach the goal (or target) term T (either explicitly or implicitly specified by a concrete term in the proof scenario or by a cost function in the compilation scenario) from the start term S directly in one big equality saturation. Rather, the process is split up into multiple stages. This corresponds to the shift from the first to the second stage in Figure 1, rectangle 2).

Equality saturation is performed until the first of the supplied guides G_1 is found. A new start term S' that matches the guide G_1 is extracted. The old e-graph is then discarded, a fresh e-graph is initialized with S' and equality saturation is performed, now trying to reach G_{n+1} or T if no more guides exist. By chaining together these smaller, more tractable equality saturations, a problem unreachable with only one

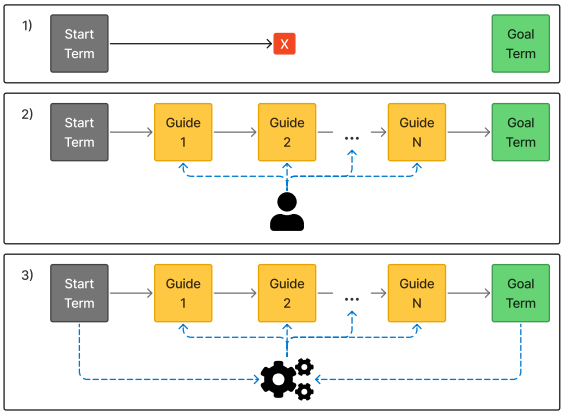

Figure 1. Extending the Scalability of Equality Saturation. A goal term not reachable in a single naive equality saturation 1) becomes feasible if an expert provides multiple guides 2), shifting some of the burden to the expert. In a *Machine Learning Guided Equality Saturation* 3), these guides are provided by a machine learning model, not requiring human experts.

equality saturation becomes tractable by reintroducing some ordering into the process [3, 9, 12].

Importantly, these guides do not have to be concrete terms and can instead be specified as precise *sketches*. Such sketches allow the user to describe the rough shape of a term without burdening them with providing the exact structure.

While this works great when an expert user can specify useful guides, such an expert is not always available. Furthermore, intricate knowledge of both the domain and the rewrite system underlying the equality saturation is required. The expert has to "guess" what might be helpful to the equality saturation, an intuition that does not always match reality. In some cases, expert guides even proved detrimental to the overall performance by leading the equality saturation down a suboptimal path [4].

3 Machine Learning Guided Equality Saturation

We propose extending *Expert Guided Equality Saturation* with a machine learning model towards a *Machine Learning Guided Equality Saturation*. This represents the final step in Figure 1 in which we replace the expert providing the guides with a machine learning model.

There are multiple possible formulations for how such a model could be built. We choose to follow the overall approach of guided equality saturation by building a model that aims to generate guides and uses equality saturation to perform the rewriting. This is an alternative to building a model that aims to predict a rewrite sequence in the enormous rewrite space.

With these guides as intermediates, we can simply reuse the expert guided equality saturation implementation. By predicting the guides, we retain a degree of interpretability in the model's output, which is not the case with other machine learning formulations. Since the actual rewrites are still performed by the underlying equality saturation engine in the e-graph, we can, by construction, never be fooled by the model into accepting a wrong proof or performing a miscompilation.

3.1 The Guide Generating Model

Our choice of model architecture is critical. While we could finetune an off-the shelve pretrained Decoder-Only Transformer architecture like LLaMa 3 [1] this approach has some key downsides: We would be required to convert the AST of the start and goal terms into a linear S-expressions and feed those into the model. This would lose all structural information and rely entirely on the model learning to recover that information from brackets alone during training. This requires a more expressive (and slower) model as well as more data compared to an model architecture with an inductive bias that takes advantage of tree structure.

At the opposite end of the spectrum, we could treat the AST as a special case of fully connected graph and use graph neural networks (GNN) methods to encode their information. We believe that this has two key drawbacks:

1. We loose inductive bias since the model cannot take advantage of the inherent tree structure of our AST. Instead, we would have to again provide the model with additional annotations to specify the position of nodes in the AST since a classic GNN architecture following the message passing paradigm can, for example, neither distinguish the order of children of a node nor its depth in a tree.
2. Research into graph generation models lags behind those established for language modeling or image generation, and there are no obvious methods to reliably generate tree structured ASTs as we require for our guides.

We decided to pursue a third alternative: Based on prior work[2, 7, 8] we designed a novel (Double-)Encoder-Decoder Tree-Transformer that takes advantage of the structural tree information. Figure 2 provides a description of this novel architecture with its two encoders for the start and the goal in the middle and on the right respectively and one decoder on the left that autoregressively builds the AST of the guide one node at a time. This architecture can be seen as an extension of both the classic Transformer architecture as proposed by [10], taking clues from graph neural networks that utilize spectral information to supply the GNN with more structural information about the graph. The key difference lies in the use of a modified disentangled attention mechanism[2]. Instead of encoding the absolute position of each node-token within a flattened linearization of the original term by adding a sinusoidal[10] or rotational[6] positional encoding to its embedding vector, we modify the attention mechanism itself to use learned positional embeddings for the relative position of a node in the AST (see Figure 3).

Thanks to the use of multi-head attention, we are able to dedicate some heads to the ancestor-descendant distance and some to the sibling position. In the future, more other distance metrics between nodes based on more advanced analysis could easily be added.

Furthermore, we modified the decoder block to contain two cross-attention layers instead of the usual single cross-attention layer, in addition to the usual self-attention layer. This allows the decoder to take attend both to the encoded start term and goal term.

3.2 Explanations as Training Data

To train the model, we require a suitably large dataset consisting of start terms, goal terms and guides in between for each pair. At the time of this writing, no ready-to-use dataset exists: We have to create such a dataset ourselves. For this, we again turn to e-graphs and equality saturation. We seed

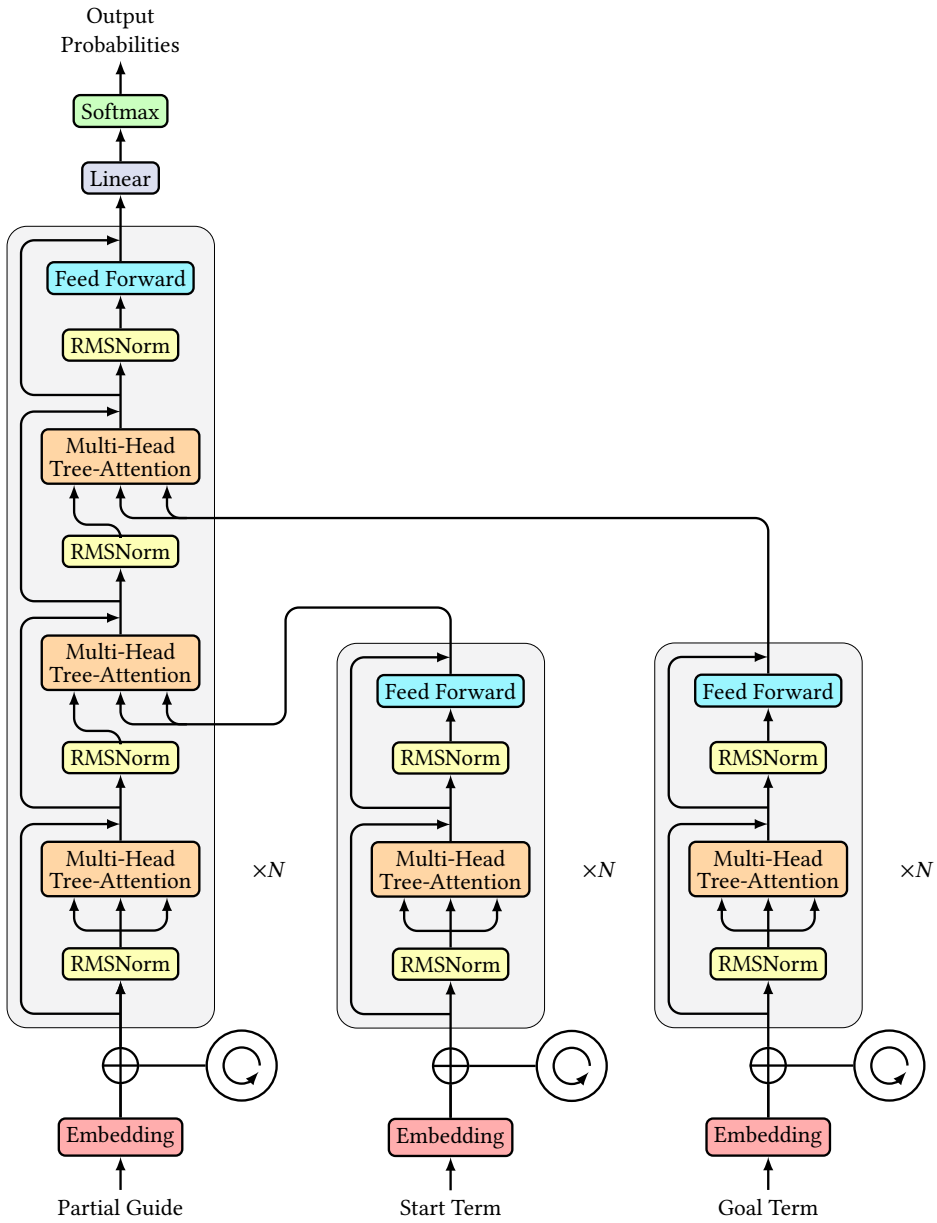

Figure 2. Double-Encoder-Decoder Tree-Transformer Architecture. The outputs of the two stacked encoder blocks for the start and the goal term are fed into the decoder block via two cross-attention layers. All Tree-Attention blocks use the modified disentangled attention described in Figure 3

the e-graph with a start term and run equality saturation with the same rewrite rules we later intend to use for the machine learning guided equality saturation until our e-graph has grown to a size of 1GB in memory. We perform an analysis to count up all terms up to double the size of the seed term present in each e-class. This allows us to sample the e-graph uniformly (up to the size limit) for terms in an e-class.

We then take the cartesian product of all sets of equivalent sampled terms and reconstruct an explanations[11], i.e. a valid sequence of rewrites from one term to the other together with their intermediate terms. While such a rewrite sequence is not guaranteed to be the shortest possible one, it provides us with a reasonably good sequence. By splitting the long sequence into shorter sub-sequences and taking the term in the middle of each sub-sequence as the guide along the rewrite path, we can finally construct the necessary (S, G, T) triples that make up our training data.

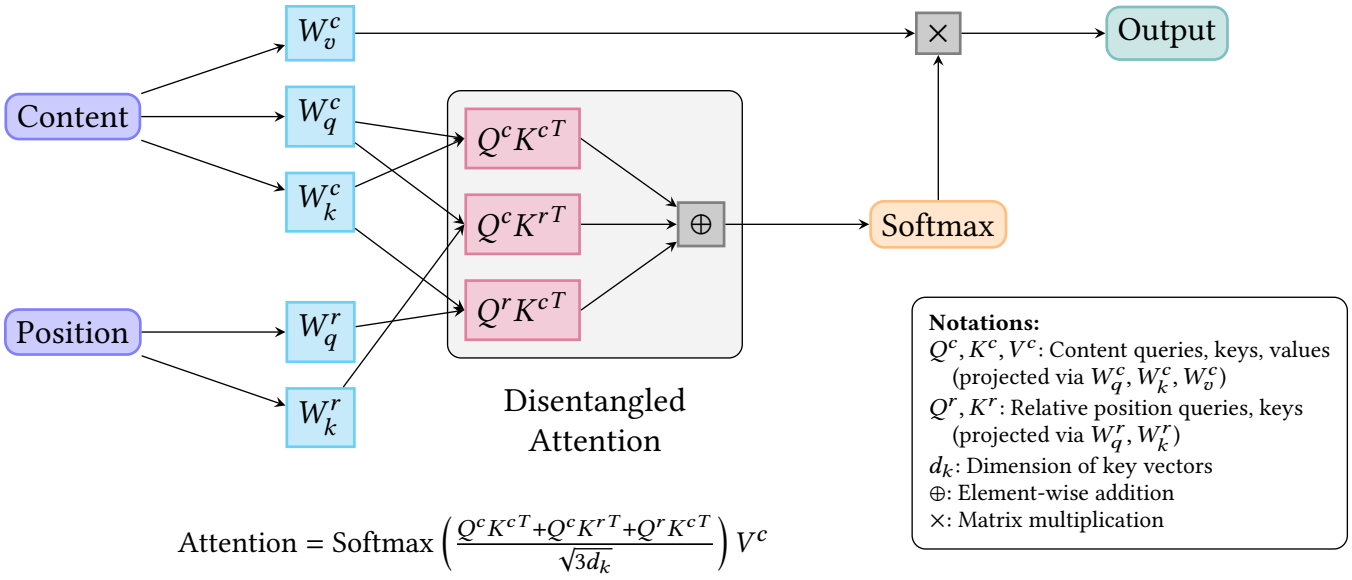

Figure 3. A single Disentangled Tree-Attention Head. Note that multiple heads are stacked together in one Multi-Headed Tree-Attention block to incorporate different distance metrics such as ancestor-descendent distance or sibling position in an AST

4 Current State

We are currently in the process of implementing and testing the model and have been able to do initial test runs with small models of 12, 92 and 312 million parameters. A falling training loss, following a typical logarithmic scaling law for both model size and training data indicates to us that the model is indeed picking up on the task at hand.

We are experimenting with how the construction of training triples from extracted rewrite sequences affects model performance. In the aforementioned test runs, we generated the triples using recursive binary partitioning. This method involves recursively splitting the original sequence into progressively smaller segments based on start and end points until reaching a cutoff of five rewrites. For each resulting subsequence, the start, midpoint, and end points are then added to the dataset. Going forward, we want to also investigate a training regime where the sequence is divided from the start into equally sized and potentially overlapping subsequences of fixed size.

In the talk, we will report on our preliminary results, specifically the first full runs of machine learning guided equality saturation and the guides generated by our model.

References

[1] Aaron Grattafiori et al. 2024. The Llama 3 Herd of Models. https://doi.org/10.48550/arXiv.2407.21783 arXiv:2407.21783 [cs].

[2] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION. https://openreview.net/forum?id=XPZIaotutsD

[3] Smail Kourta, Adel Abderahmane Namani, Fatima Benbouzid-Si Tayeb, Kim Hazelwood, Chris Cummins, Hugh Leather, and Riyadh Baghdadi. 2022. Caviar: an e-graph based TRS for automatic code optimization. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction (CC 2022)*. Association for Computing Machinery, New York, NY, USA, 54–64. https://doi.org/10.1145/3497776.3517781

[4] Thomas Kœhler, Andrés Goens, Siddharth Bhat, Tobias Grosser, Phil Trinder, and Michel Steuwer. 2024. Guided Equality Saturation. *Proceedings of the ACM on Programming Languages* 8, POPL (Jan. 2024), 58:1727–58:1758. https://doi.org/10.1145/3632900

[5] Rudi Schneider, Marcus Rossel, Amir Shaikhha, Andrés Goens, Thomas Kœhler, and Michel Steuwer. 2025. Slotted E-Graphs. *Proc. ACM Program. Lang.* 9, PLDI (2025).

[6] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. RoFormer: Enhanced Transformer with Rotary Position Embedding. https://doi.org/10.48550/arXiv.2104.09864 arXiv:2104.09864 [cs].

[7] Ze Tang, Chuanyi Li, Jidong Ge, Xiaoyu Shen, Zheling Zhu, and Bin Luo. 2021. AST-Transformer: Encoding Abstract Syntax Trees Efficiently for Code Summarization. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1193–1195. https://doi.org/10.1109/ASE51524.2021.9678882 ISSN: 2643-1572.

[8] Ze Tang, Xiaoyu Shen, Chuanyi Li, Jidong Ge, Liguo Huang, Zhelin Zhu, and Bin Luo. 2022. AST-trans: code summarization with efficient tree-structured attention. In *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 150–162. https://doi.org/10.1145/3510003.3510224

[9] Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. 2021. Vectorization for digital signal processors via

equality saturation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 874–886. https://doi.org/10.1145/3445814.3446707

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[11] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. egg: Fast and extensible equality saturation. *Artifact for "Fast and Extensible Equality Saturation"* 5, POPL (Jan. 2021), 23:1–23:29. https://doi.org/10.1145/3434304

[12] Yichen Yang, Phitchaya Phothilimthana, Yisu Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. 2021. Equality Saturation for Tensor Graph Superoptimization. *Proceedings of Machine Learning and Systems* 3 (March 2021), 255–268. https://proceedings.mlsys.org/paper_files/paper/2021/hash/cc427d934a7f6c0663e5923f49eba531-Abstract.html