

DALLIN DAHL

dallinjdahl@gmail.com | github.com/dallinjdahl

RESEARCH STATEMENT

As global supply chains expand, there is an increasing need to ensure the security and reliability of our computing stack. This assurance needs to be approached from both the engineering and reverse-engineering sides. My work has focused on applying deep learning and graph theory to reverse engineer IP from unlabelled circuits. Although reverse engineering can be used for malicious purposes, it also enables theft detection, trojan detection, and toolchain verification.

We look at IP detection as a node-classification problem on a graph, labelling each node (circuit element) as part of or separate from the IP. Our initial research framed this process as a neighborhood recognition problem, where each node was labelled if its neighborhood “looked like” one we had seen in the IP. That insight led to the development of a novel algorithm with best-in-class performance characteristics (memory and runtime linear in size of training data and target), while still maintaining high levels of accuracy (F1 of 80% on a Microblaze soft processor). However, we discovered that toolchain optimizations and configuration options significantly changed the neighborhood and degraded the accuracy of our algorithm.

The remainder of our work framed this node-classification as a generalized object-detection problem analogous to object detection in computer vision. We employed Graph Neural Networks (GNNs) to do the node labelling, significantly increasing our accuracy and robustness (80–99% accuracy across 1000 designs). This also involved building a tool to generate random-yet-plausible circuits to train the GNNs, resulting in several datasets of 5000 designs, all run through the entire EDA toolchain.

The stochastic nature of deep learning motivated us to quantify how robust our models are to changes in the random initialization, graph structure, training distribution, and toolchain version. This culminated in a Bayesian model illustrating that while GNNs are robust to changes in the random initialization, some architectures struggle with certain graph features like high fanout and buffer nodes.

In the future, I want to explore how we can decompose post-implementation netlists into discrete hierarchical modules. Rent’s rule describes the observation that for designed circuits there is a power-law relationship between the number of terminals in a circuit and the number of gates in that circuit. This indicates that human designers leave a footprint on the graph structure of the resulting circuit. Previous work has focused on finding the best decomposition of a given circuit so as to closely match Rent’s rule at every partition. These efforts often rely on hand-crafted heuristics and don’t transfer across graphs. Applying GNNs and a large and varied corpus of circuits beyond that already developed, we can train a model to recognize that footprint across a range of circuits. IP detection becomes a labelling problem, finding hierarchical levels of the right size and matching them to known IP.

In addition to netlist decomposition, I'd like to expand on my work with Bayesian analysis. Previous work with explanatory Item Response Theory (IRT) has explored how various features can impact the results of deep learning models in general, but has yet to extend to graph-structured problems. Our work on explaining the failure modes of the GNNs on netlist-level IP recognition naturally extends towards a full explanatory IRT style framework for analyzing GNNs with an eye specifically towards the features that make GNNs difficult: graph-theoretic features and training artifacts including preprocessing of graphs and toolchains. This methodology could easily expand to other graph-structured domains like call and dependency graphs in software. The result of this research would be a framework and a tool to analyze classifiers across graph-structured data, providing insights on both the classifiers and the data.

Complementing our efforts in reverse engineering, I also want to approach FPGA security by construction. Most trojans either work by leaking data to an output or sabotaging the design with a kill switch. I want to design a hardware description language and associated toolchain that enables proof-carrying hardware for confidentiality and liveness. There exist other works in verifying designs, but they typically don't propagate through the toolchain. Proof-carrying hardware work, which has explored propagating the guarantees through the toolchain, typically involves working in either a proof-assistant or model-checking framework, with the extra overhead and limited scope those imply (liveness is specifically out of scope). This work would target properties like liveness and confidentiality, maintained through the entire toolchain and tractably checkable post-hoc. Linear/quantitative type systems can ensure properties of confidentiality, while productivity of codata ensures liveness for sequential logic. Combinational loop detection is a trivial example of a guarantee that has been checked at the RTL level, but not pushed through the entire toolchain.

In the near future, I want to expand the focus of these techniques to include several layers of the computing stack. Graph-theoretic and deep learning techniques are extremely powerful tools, and apart from the hardware supply chain, our software supply chain is at an even greater risk for intrusion. Being able to ensure the reliability and security of the whole stack means assuring not just the hardware and software separately, but also their interactions, especially with respect to the hardware-accelerated software flows prevalent in the AI world. Eventually, I want to establish a program focused on applying deterministic and probabilistic methods to establishing the security and reliability of computation. These deterministic methods include graph theory and extend into static analysis, proof search, and type systems.