We thank all reviewers for their valuable comments. We address the concerns raised by them below.

Common Concerns

Novelty

Reviewer-2 *The novelty is incremental. This paper directly combines well-known techniques and does not make any new contribution from the machine learning perspective.* The main contributions of our work is to show how to reduce compiler auto-vectorization to a sequential decision making problem and how to solve it by learning a gated graph neural network (GGNN) based policy using imitation learning. Based on our knowledge, this is the first time a compiler pass is learnt end-to-end from data. Further, in our evaluation, we discuss how different traversal orders affect the efficacy of the learnt policy, which gives insights to designing imitation or reinforcement learning algorithms in the compiler / program transformation domain. Hence, we consider that our work is a valuable addition to NeurIPS under the applications track.

Reviewer-1 *The idea of using imitation learning to make approximate decisions is not new.* Imitation learning has been successfully demonstrated to work in autonomous driving etc. Based on our knowledge, there has been limited work on successfully applying it to discrete decision making problems. In fact, optimal subset selection problem reduces to compiler auto-vectorization (Section 1), which is known to be NP-hard. We are using imitation learning to learn a policy to approximately solve this NP-hard problem.

Experimental Evaluation

Reviewer-1 *The experiments are superficial. There is no comparison of the actual compile time and execution time with existing methods.* **Reviewer-2** *.... The author needs to provide a wall-clock time cost comparison of different methods.* The main purpose of our work is to show that an end-to-end learnt policy guided by optimal decisions can outperform traditional compiler heuristics. We evaluate the policy on real world programs (Figure 3 and 4) and show that we are outperforming LLVM handily in terms of its own cost model. We expect this to translate to actual execution times, since the cost model was designed to reflect how fast code would run on real hardware. Integrating the learnt policy inside the compiler does not involve further research, but however is a considerable engineering challenge, which we are currently undertaking and is necessary to get end-to-end execution times. We will include them in the final verision. Nonetheless, our results clearly show that we learn a better policy than the heuristics used by LLVM.

Case Studies

Reviewer-2 *The author should provide more detailed experiment results, such as case studies of some packing strategies.* **Reviewer-3** *What are corner cases that LLVM does not optimize, but this solution does?* One interesting case where the learnt policy outperforms LLVM's heuristics is that it prefers to vectorize floating point divisions even with high packing and unpacking overhead. When we individually time the functions, we find indeed this is beneficial, since floating point division is an expensive operation and doing it in parallel reduces execution time. The learnt policy picked up this fact entirely from data. Further, there are other more involved patterns which the learnt policy prefers to vectorize which we will include in the final version.

Reviewer Specific Questions

Reviewer-1 *Auto-vectorization is an important problem in the compiler area. The paper should be more suitable to compiler/parallelization conferences such as PACT or CGO.* We came up with a novel GGNN formulation that allows us to learn a policy to select profitable instruction packing opportunities. Additionally, we discuss how different traversal orders of the GGNN formulation affects the overall policy. This gives insights on how to design imitation or reinforcement learning algorithms in the compiler / program transformation domain. Taking all this into account, we consider our work a self-contained solution that is worthy of publishing at NeurIPS in the applications track.

Reviewer-2 *... no mathematical description of the used algorithms such as GGNN and DAGGER...*
We will include the mathematical formulations for GGNN and the DAGGER algorithm in the final version of the paper.
The author should show how transferable the learned agent is. The learnt model is transferable to programs in the hidden test set, which consists of well-known benchmark programs from the SPEC benchmark suites.

Reviewer-3 *Extend to cases where vector operations are not necessarily preferential.... by learning from real data or building a model of the hardware.* This is indeed an interesting future direction which we are planning to pursue.
does depth of the neural network/e.g. message-passing steps matter? We find that the residual connections in the GGNN formulation is important to ensure we learn a generalizable policy. We will discuss about this more in the final version. The depth of the neural network does not matter that much.
why is the model inferior to the optimal solution? We are imitating an optimal solution, therefore the model is upper bounded by the optimal solution and hence it should be inferior. However, we are investigating the gap between the optimal solution and the learnt policy and ways to reduce it.
Would more data or a higher capacity model help? We hypothesize more data would help learning and generalizability.