Approximating Functions for Grid Domains

Danny Zhu Yuzi Nakamura Ben Humberston

Approximate computing sacrifices strict computational accuracy to reduce execution time and energy usage. Many problems in AI, graphics, and image processing do not require bit-level exactness in computation in order to produce acceptable results. For these cases, the savings in battery life, runtime, or power costs may be worth more than the loss in accuracy.

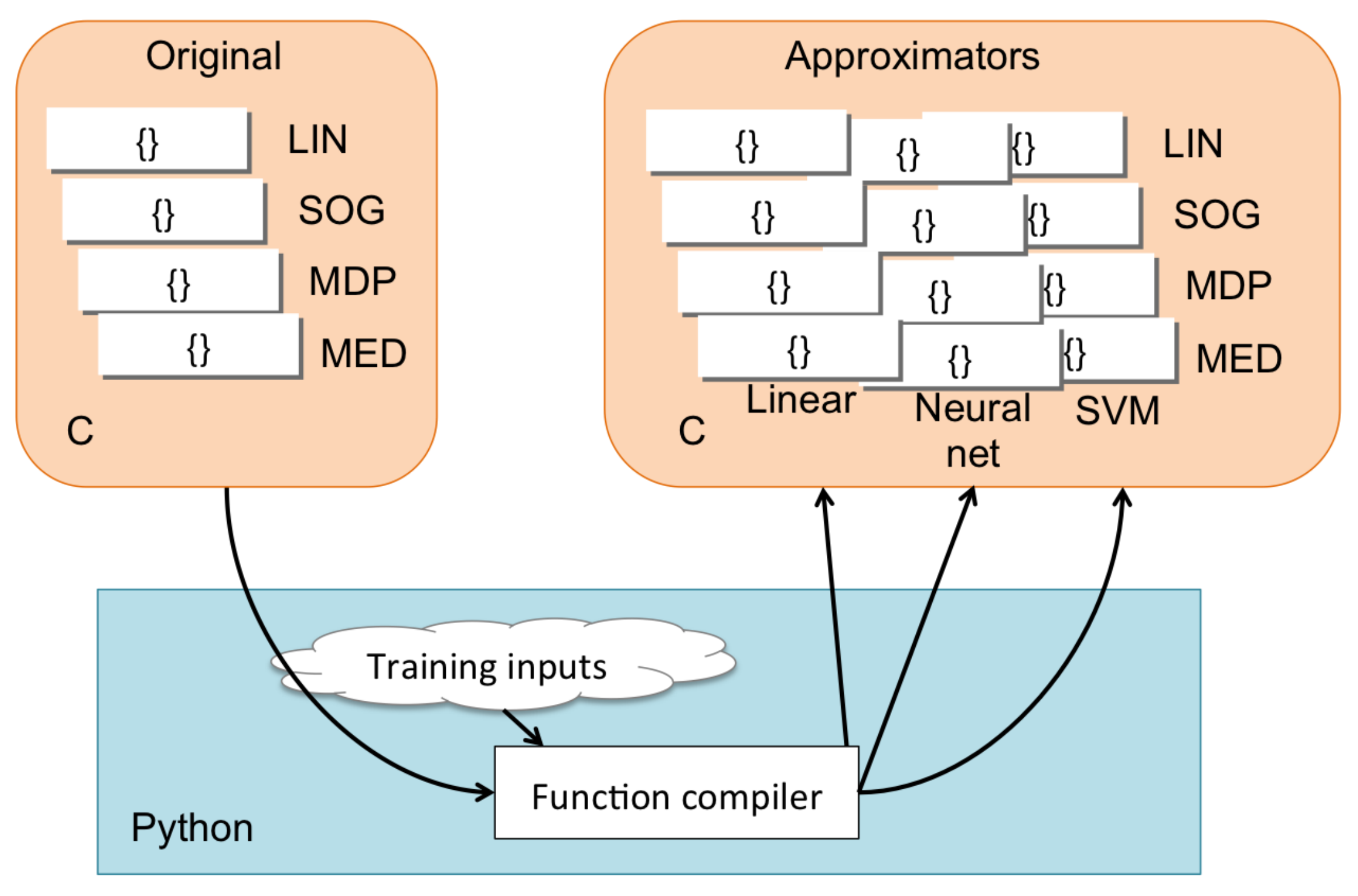
Approximate compilation automatically converts input source functions into approximated versions based on annotations by the programmer of which areas are deemed approximable.

In this project, we take C functions that output discrete 2D grids arrays and compile them into several different approximated versions. We analyze the approximations to determine the speedup and how well they match the original grid output for a number of possible function types and inputs.

Method

The structure of our compiler is shown to the right. The user specifies a function for approximation in a C source file as well as a set of typical inputs to the function. The compiler generates an approximate version of the function by training on those inputs and their associated grid outputs.

The generated source of the approximate version of the function is suitable for drop-in replacement into the original code base.



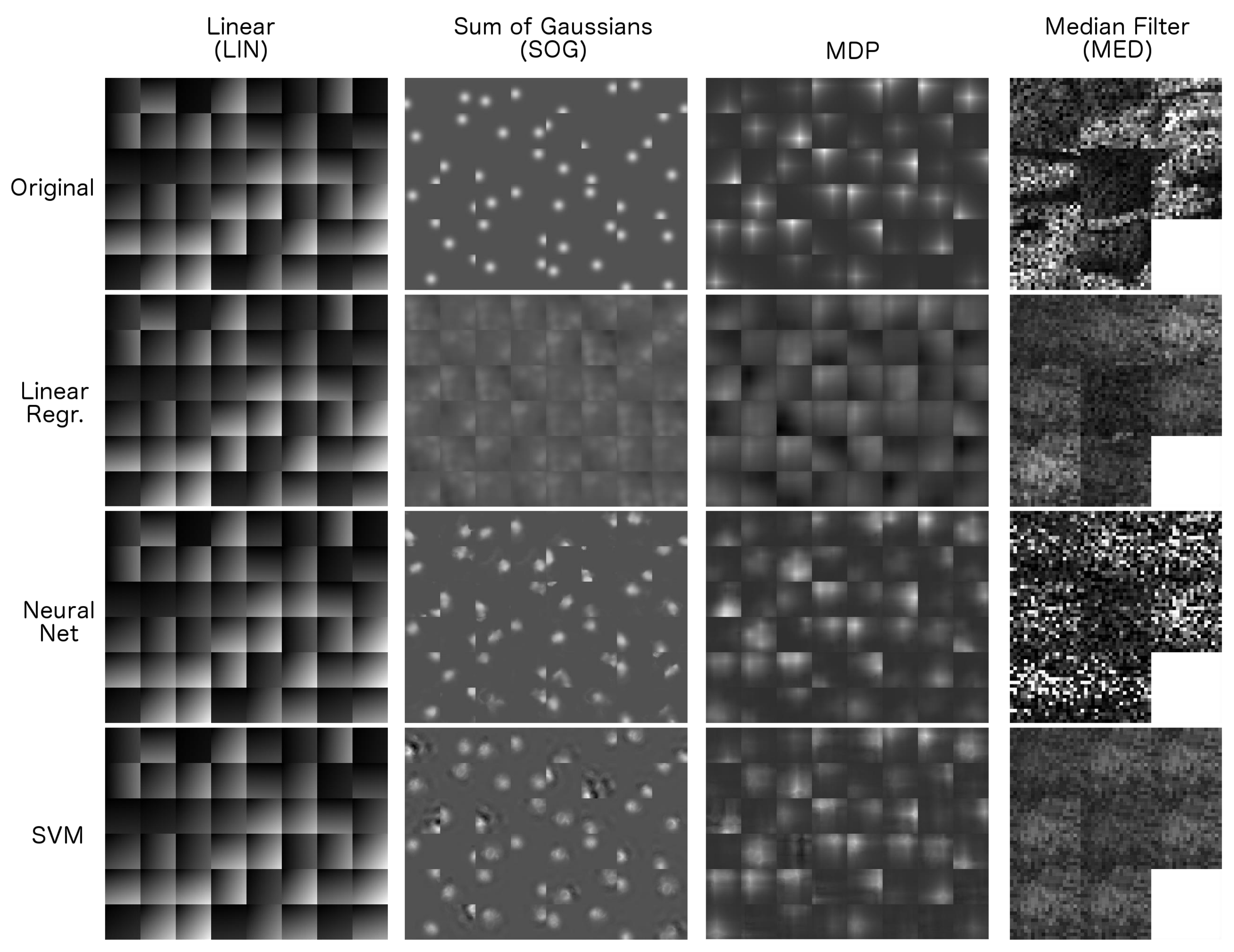
Evaluation

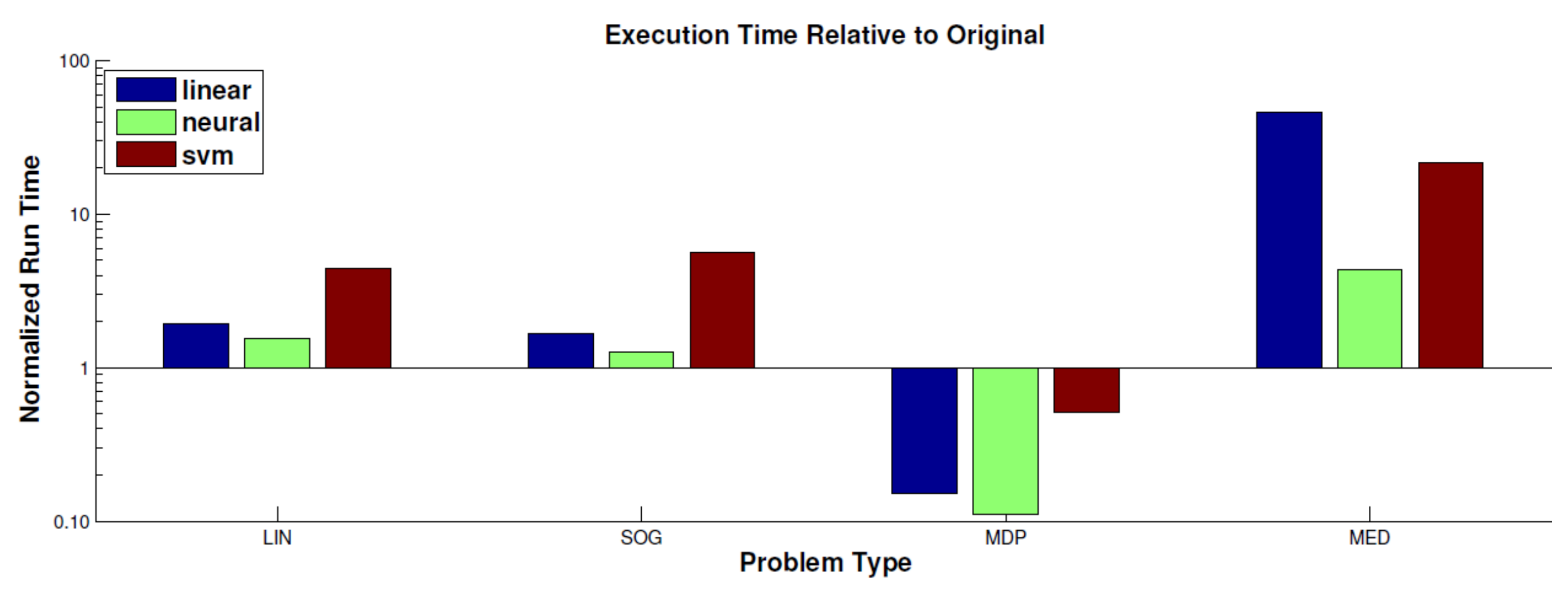
Our compiler allows for different approximation strategies via a generic interface. For this project, we implemented three types: **Linear Regression, Neural Nets,** and **SVMs**.

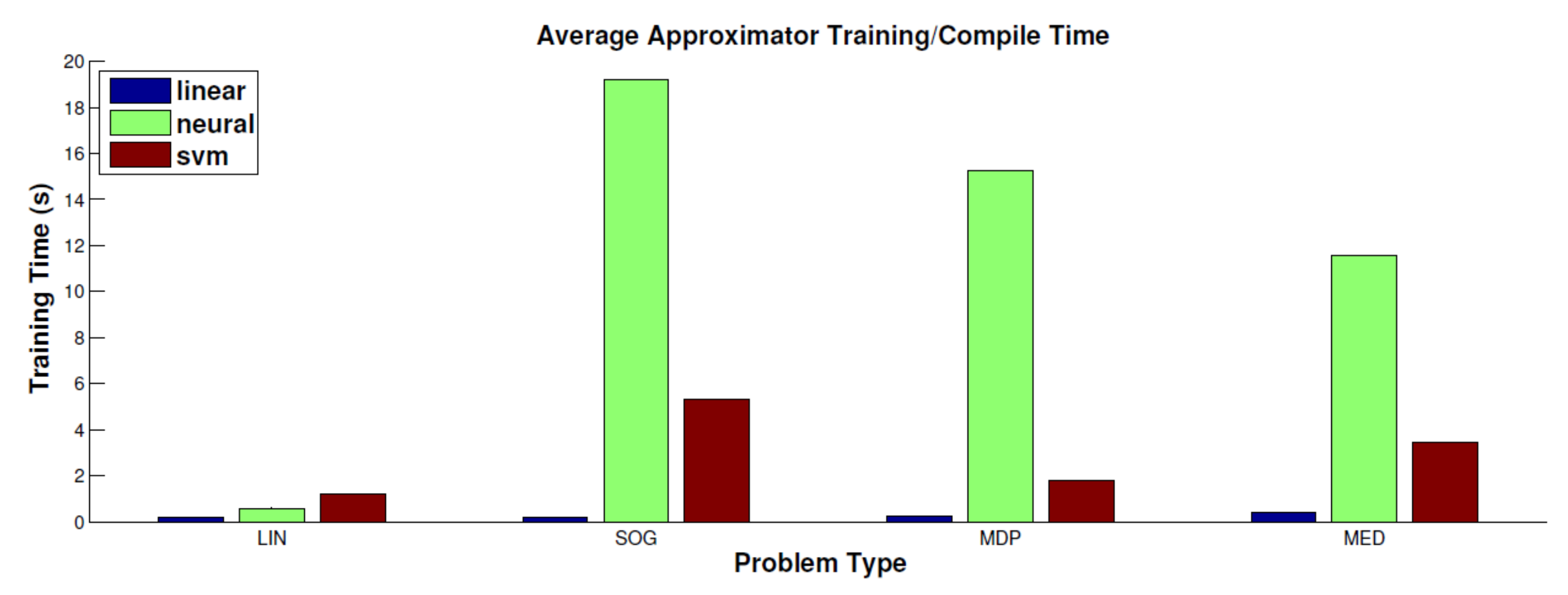
We test each of these approximation types on four input problem domains that are amenable to approximation:

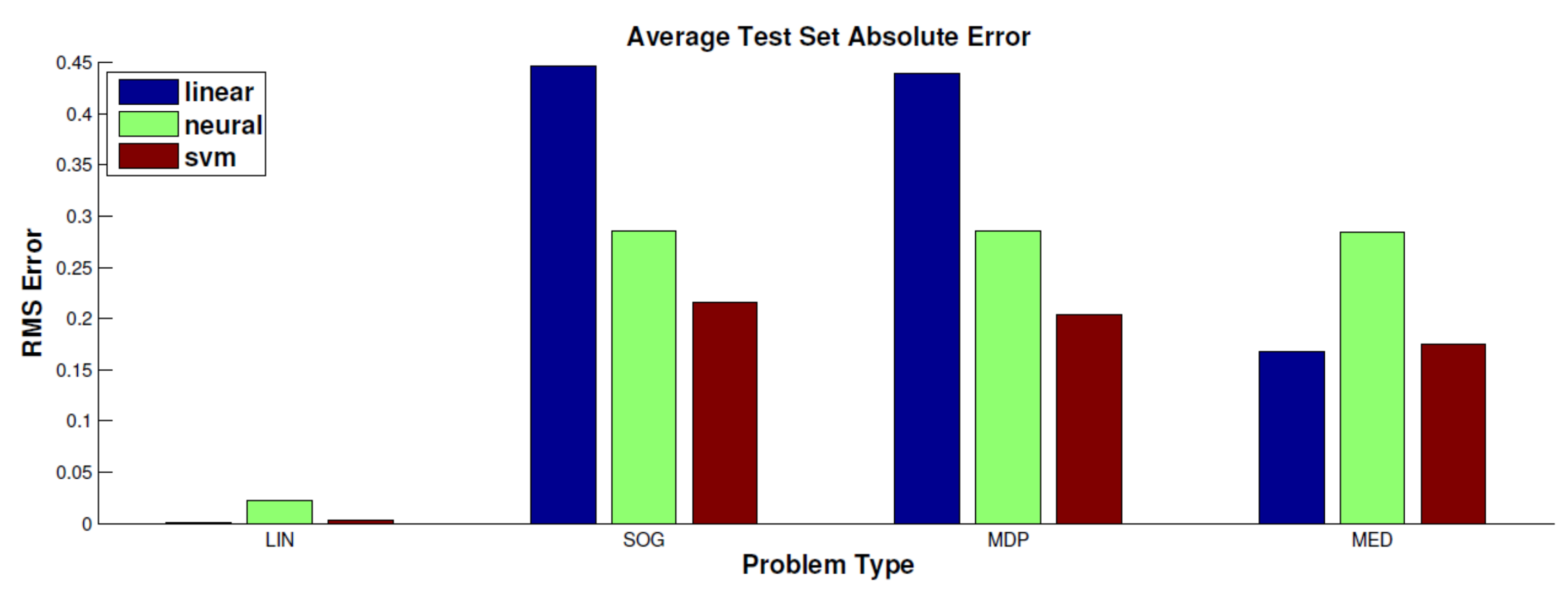
LIN: Linear gradient field (baseline test)
SOG: Sum of Gaussian bases over a 2D field
MDP: Value iteration for a Markov decision process
MED: Median value filter applied to image patches

Examples of these problem types are shown to the right, along with the approximated outputs.









Results

As expected, all approximations performed well on the baseline **LIN** problem domain. In the more difficult **SOG** and **MDP** domains, the neural net and SVM approaches have the advantage over the linear regression approximation, which matches the qualitative impressions prompted by the example outputs above.

While the neural net approximations tend to have long training periods and slightly higher errors than SVM approximations, they also execute in less time. A user may prefer one or the other based on how he/she prioritizes these traits.

Similar to [1], we found that the approximations only yielded faster performance than the original functions in very computationally expensive problem domains (**MDP**).

As such, future work in approximate compilers may further explore the potential of specialized hardware such as NPUs (neural processing units) to accelerate approximate code execution.

[1] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-45, (Washington, DC, USA), pp. 449–460, IEEE Computer Society, 2012