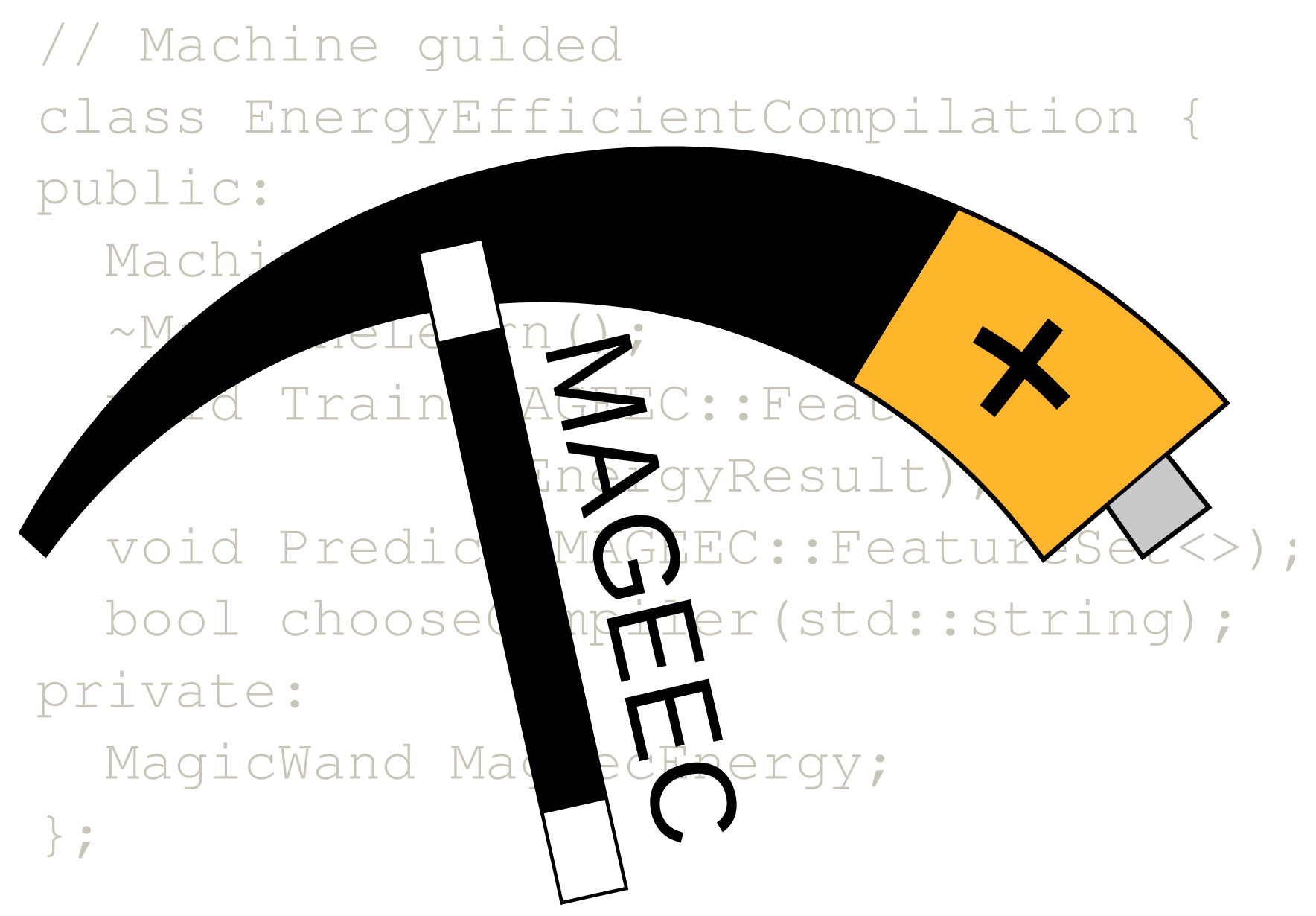
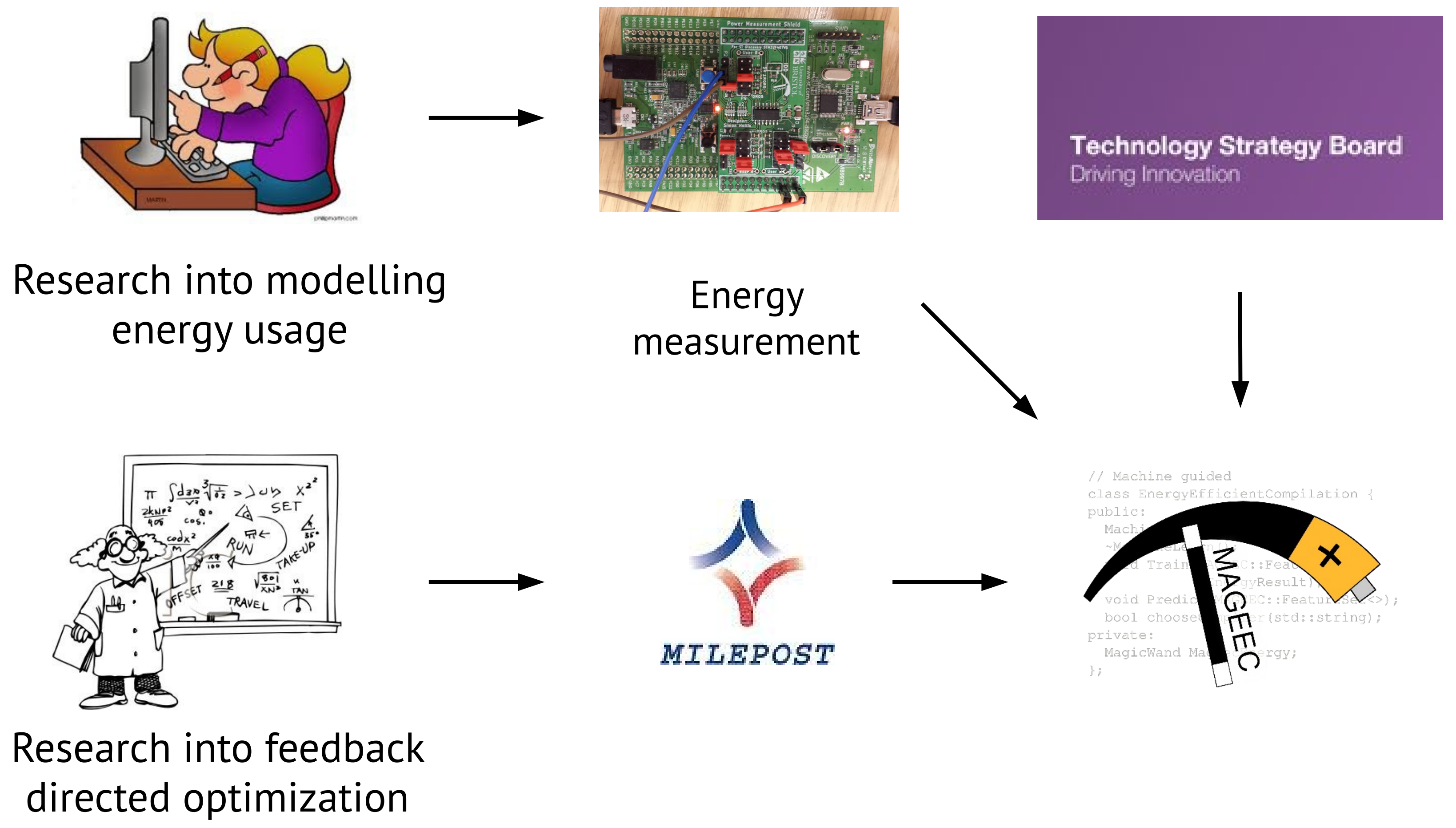

// Machine guided
class EnergyEfficientCompilation {
public:
void Predict(MAGEEC::FeatureSet<>);
bool chooseCompiler(std::string);
private:
};
MAGEEC

Machine Guided Energy Efficient Compilation

Using machine learning to select compiler optimizations that minimize energy consumption

How We Got Here



Research into modelling energy usage

Energy measurement

Technology Strategy Board
Driving Innovation

Research into feedback directed optimization

MILEPOST

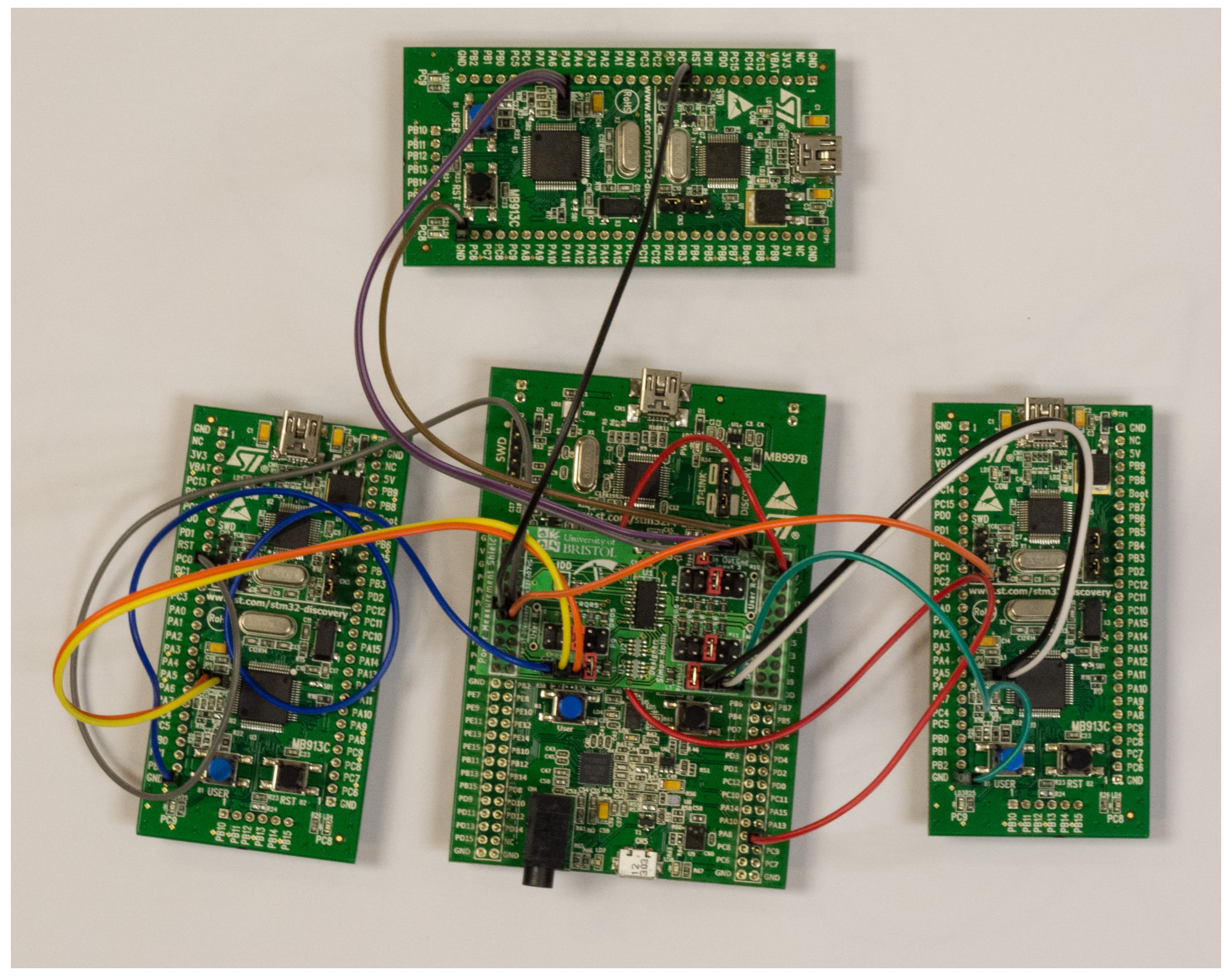
MAGEEC

Effects of Selecting Passes

Benchmark	% improvement	
	Iterative elimination	Modified version
2dfir	8.3	8.3
blowfish	1.5	1.5
crc32	0.7	1.0
fdct	11.0	15.3
float matmult	1.9	2.1
int matmult	4.6	4.7
sha	0.3	0.3

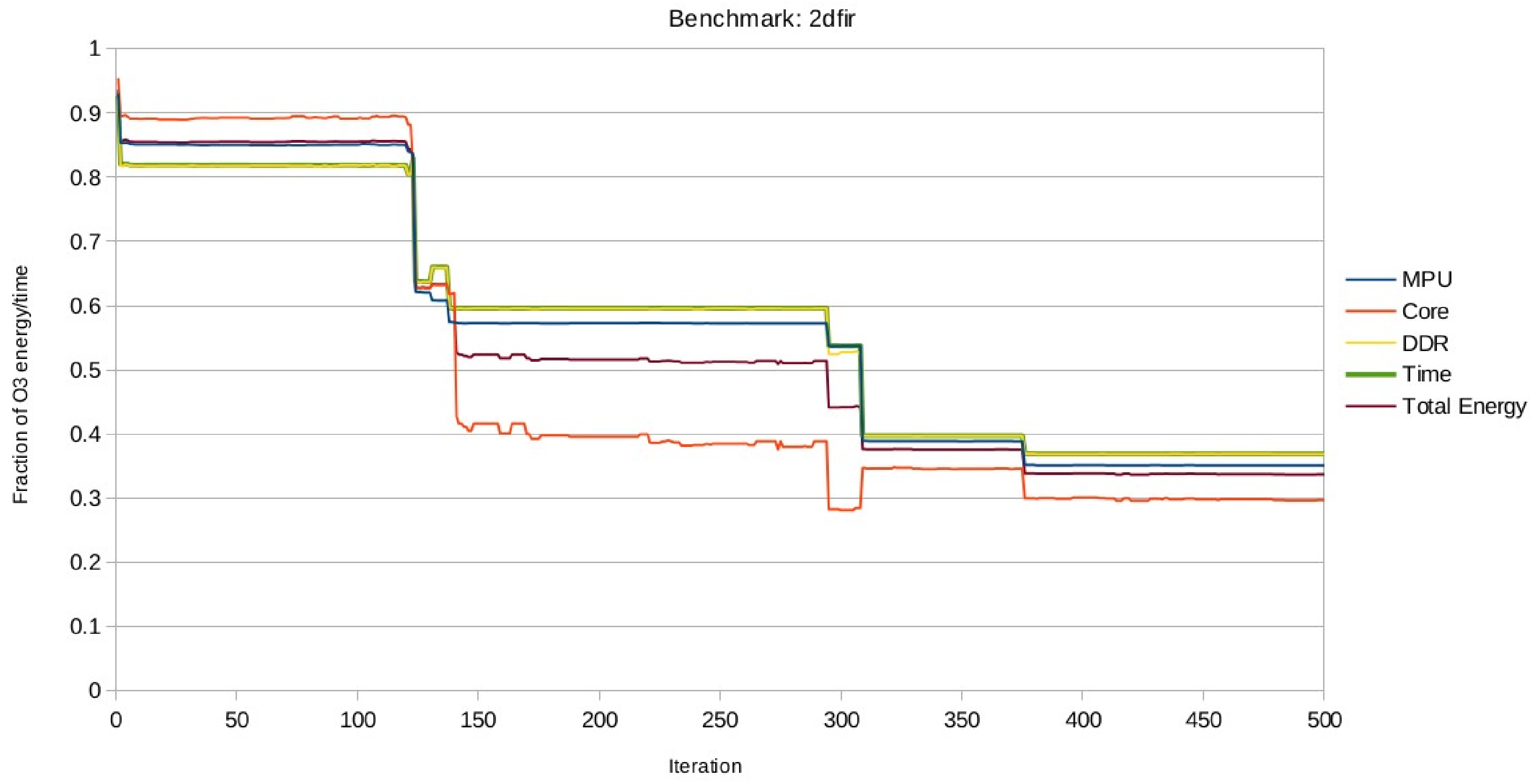
In this example, iterative compilation was used to select the optimal passes that should be used to compile a program. This shows that compared to -O3 improvements of up to 15% can be made.

Energy Measurement Hardware



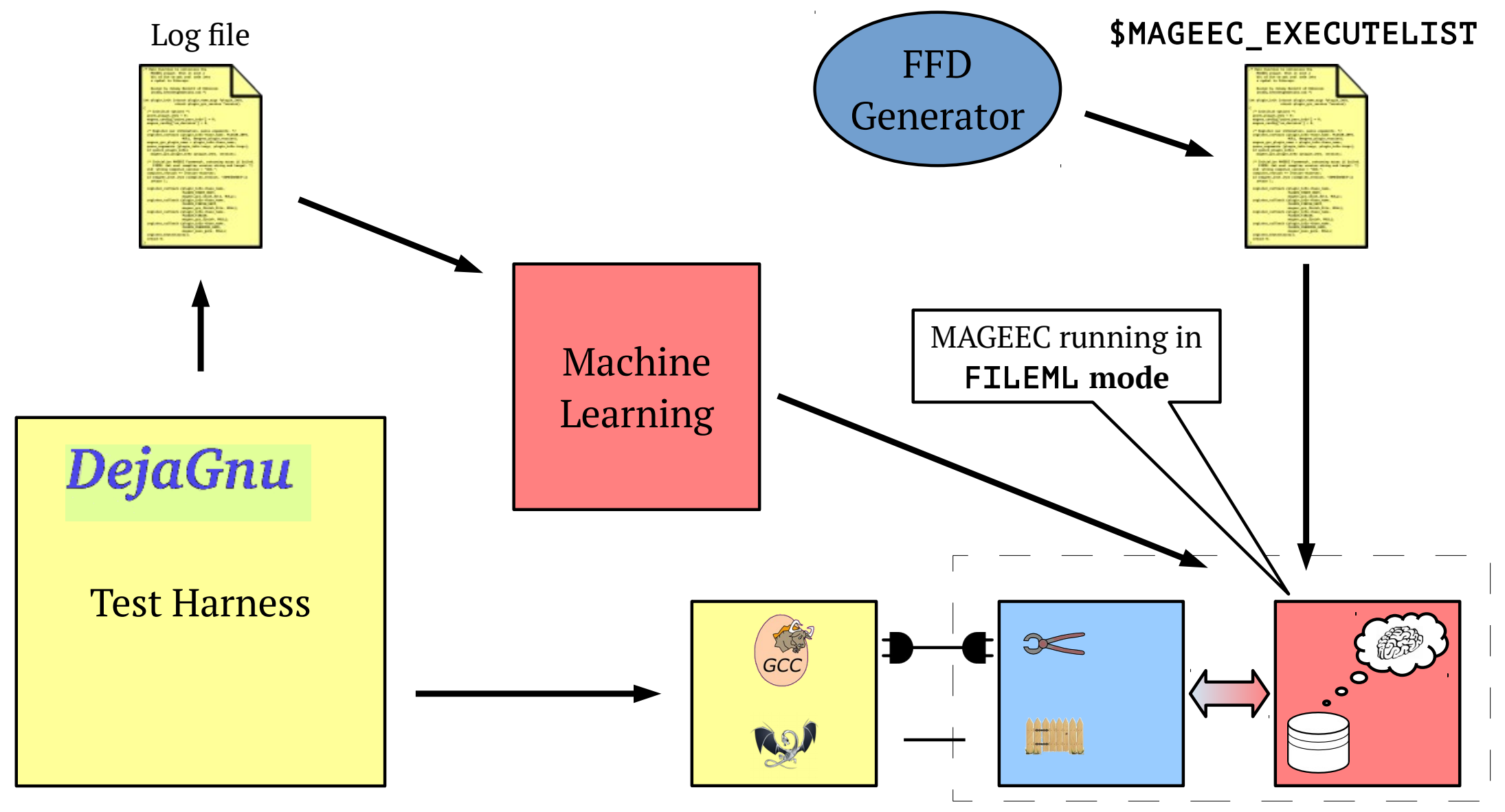
For this project we developed energy measurement hardware capable of measuring voltage and current up to 2 million samples a second with an accuracy of better than 1%. We use this to measure energy consumption of the various compiled programs.

Effects of Reordering Passes



In this example, genetic algorithms were used to select optimizations to run, with the aim of minimizing energy. Over 500 generations energy was reduced by 65% with a time reduction of 60% compared to clang's -O3.

Training for LLVM



Log file

FFD Generator

$MAGEEC_EXECUTELIST

Machine Learning

MAGEEC running in FILEML mode

DejaGnu

Test Harness

MAGEEC is trained using BEEBS (www.beebs.eu), an open source benchmark suite optimized for deeply embedded devices.

To train we use a “clang-like” driver which uses *opt* for fine grained control over passes. We record flags, passes and a feature vector which identify the program. After results have been gathered and normalized, we provide the machine learner with the “best” pass combination for each program.

At run-time we switch to clang and use the machine learner to decide which passes should run to produce the best executable.

Future Work

a) Usability – having an “-Oe” flag.

b) Improve MAGEEC's ability to live outside the compiler, which will require development of a plugin API for LLVM.

c) Add knowledge of pass dependencies to training flow.

d) In addition to disabling passes, allow pass reordering and re-execution.

e) Experiment with other machine learners and data sets. It is not clear that the current choice (decision trees) is best for this problem.

f) Explore other possible (multi-objective) optimization criteria. Candidates include code size, execution speed, run-time memory usage, build speed and build energy.

EMBECOSM®

mageec.org github.com/mageec beebs.eu

University of BRISTOL