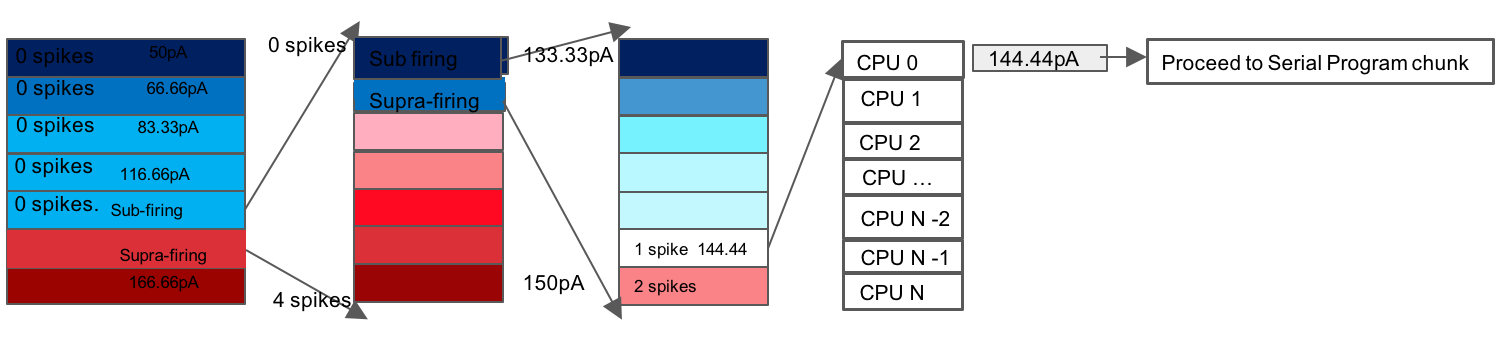
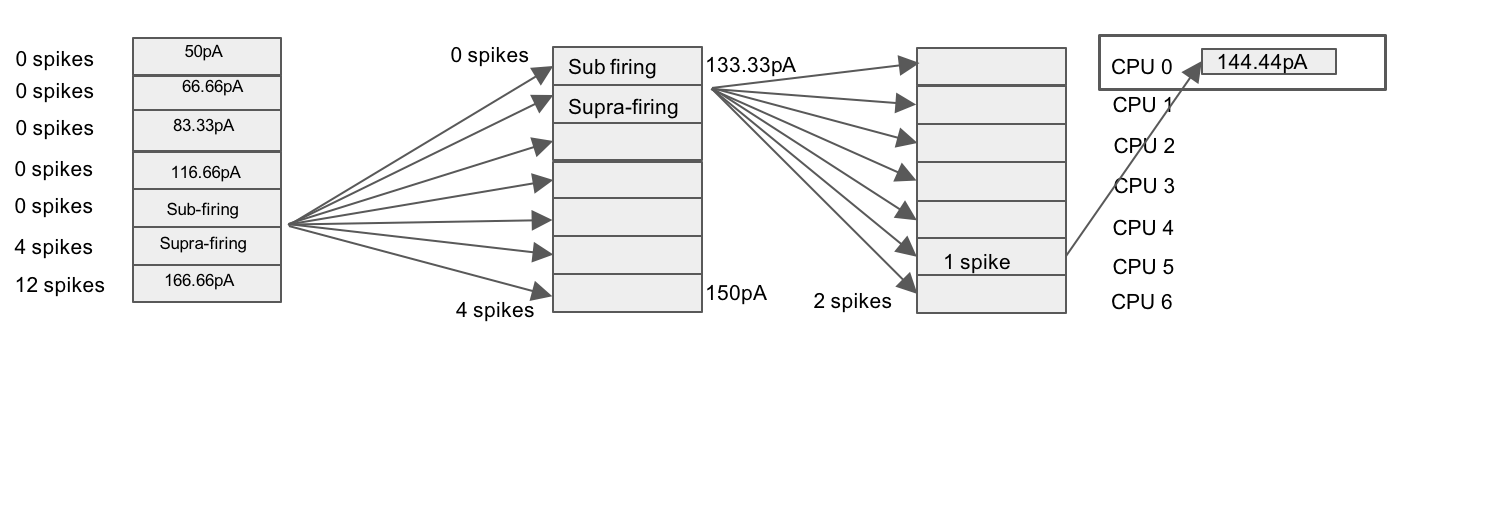


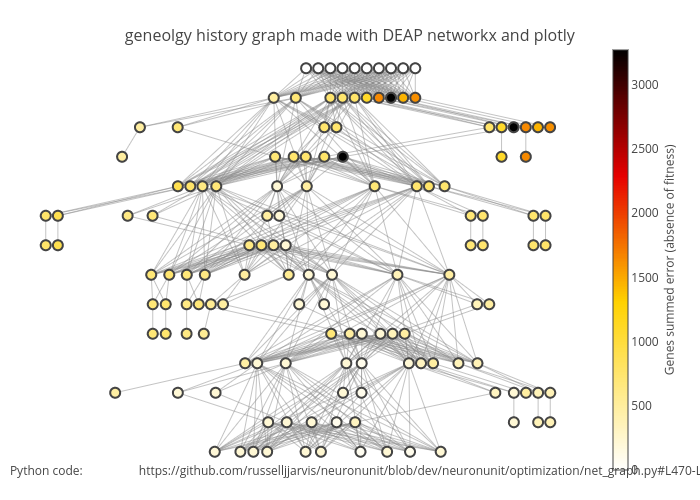
*Figure 1*: Illustrates the function of the optimization algorithm. The algorithm consists of code blocks dedicated to different processing functions. The first function performed is to explore differences in model behavior implied by neuronal model parameters encoded by the genes of the genetic algorithm. Another major code block function is responsible for evaluating the fitness of output model behavior resulting from particular gene states. The evaluation of a fitness function was performed inside the *NeuronUnit* test suite. Importantly the algorithm scales by spreading workload onto un-utilized CPUs. Code blocks that are implemented with parallel map functions are depicted in the diagram using depth and diverging arrows, and in the case of the Rheobase search, in its own separate diagram. Parallel scaling of the program workload occurs at two points of high computational workload, where invocation of a neuron simulator occurs. Both of these points act as bottle necks, since all subsequent processing is dependent on, and must wait for these invocations to return. Failing to spread out the workload would cause severe performance degradation, and thus delegating out work in an embarrassingly parallel fashion, facilitates timeliness of the optimizers convergence.





Workers [1,6] are momentarily unloaded, the rheobase value gets collated with other model parameter values on Worker 0, and then either the pattern repeats over un evaluated models, or the program moves on to performing processing intrinsic to other functions.

*Figure 2*. As described in figure 1, there were several computationally intensive sites in the program workflow, chief among these sites was the search for threshold current injection value which varies between models and must be performed on a per model basis. Many of the NeuronUnit test criteria depend on a threshold current value for the model being known in order to render the tests viable. The search for a threshold current injection value, ie a current amplitude over 100ms that results in only one neuronal spike, was speed up using brute force. Intervals containing a plausible range of current injection values were at first coarsely sampled across 7 CPUs, however after the workers return from this coarse grained sampling, their results inform a narrower interval to search, this narrower interval is then able to be sampled at a much finer level of granularity, and the pattern continues until a threshold current injection value is found.



*Figure 3*

The DEAP library and other python libraries such as *networkx* *plotly* enabled a visualization of changes in model fitness that result from gene evolution progress. This diagram serves as reflection of the genetic algorithms performance, as it illustrates how unfit models are rapidly discarded, and that error is minimized over relatively few generations. This visualization also provides educational value about the workings of Genetic Algorithms by clarifying the relationship between model fitness genetic recombination and inheritance.

*Figure 4*

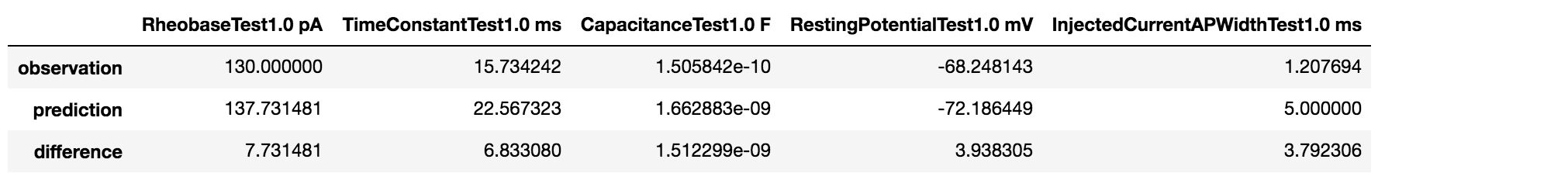
This figure demonstrates the large diversity of model behavior was sampled and appropriately scored by the combined action of *NeuronUnit* and *DEAP*. Models exhibiting Z-scores that were the furtherest from zero were deemed fitter. As figure 1 illustrates before the Genetic algorithm is initialized appropriate *NeuronUnit* tests are constructed based on cell specific data. As a proof of concept we used the optimization routine to fit the Izhikevitch model to a experimentally derived Neocortex pyramidal cell layer 5-6 electrical characteristics. This figure shows how models that had a resting membrane potential close to -68.24 *mV* were assigned a higher fitness value. Other important wave form features that are also visible in this figure include variation in spike height, and spike length. Each membrane potential in the figure, has exactly one spike, the result of applying a model specific rheobase current injection value, over the duration of *100ms*



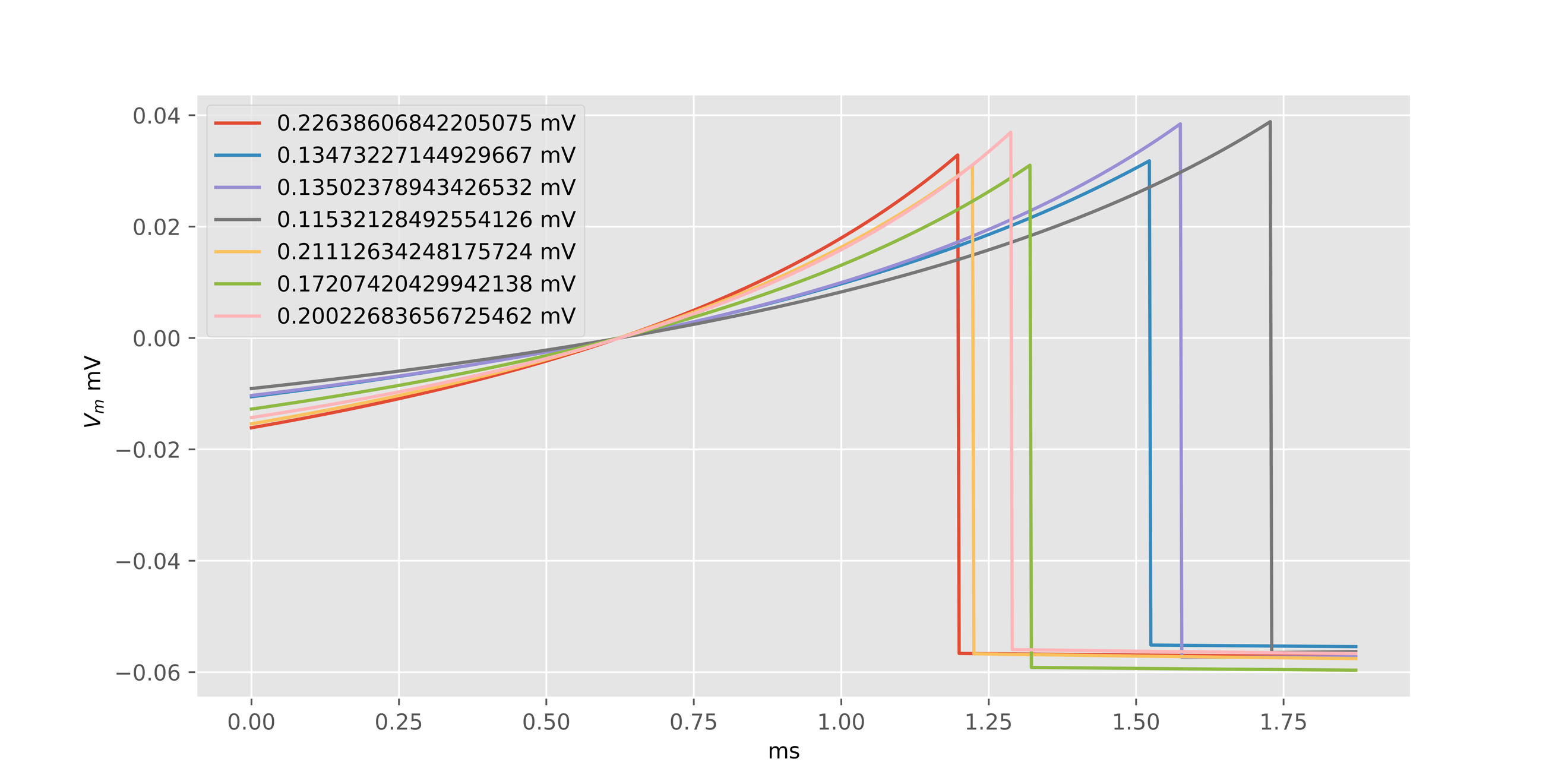
*Figure 5* shows an error surface corresponding to parameter a and parameter b, in the Izhiketich neuronal model. In this case existing code for visualizing the error surface was leveraged from a necessarily different, yet overlapping optimization effort *BluePyOpt*. In the context of a regular exhaustive search process, the effects of varying parameters are systematically explored, where one variable is held constant while the other variable increments. However the error surface depicted in *figure 5* was acquired differently: rather than holding eight out of ten Izhikivitch model parameters constant, while the other two are systematically varied, this plane into the error surface instead depicts a cross section into an error hypervolume, from the perspective of sampling taking place inside the Genetic Algorithm, where each parameter is varied simultaneously, and resulting errors in the row and column matrix elements, are not exclusively caused by changes in parameters a, and b. Inspection of error planes sampled directly from inside the Genetic Algorithm reveals complex surfaces with multiple minima. These complex multiple minima surfaces warrant use of a more robust search strategy over less robust gradient descent based strategies. The Genetic Algorithm is more robust to the effects of local minima, since mutations added after genetic recombination help to preference surface exploration without compromising solution quality.

This figure belongs near the text.

For a single model class and biological neuron type, we obtained an error surface via a Non-Dominated Sort Genetic Algorithm (NSGA), corresponding to plausible subsets of candidate parameter values that respect intrinsic biological diversity.



*Figure 6*: Optimizer results, and observation versus prediction agreement. Five out of eight optimizer results showed good agreement between experimental observations and model predictions, corresponding to the best candidate solutions produced by the previously described genetic evolution process. Importantly there are some expected differences between observations and predictions, and these minor variations may actually be some what favorable reflections of important biological diversity.



*Figure 7* and *8 NeuronUnit* provides many convenience methods that interface experimental recording values to theoretical model predictions, Additionally however *NeuronUnit* includes useful feature extraction algorithms relevant to spiking neuronal waveforms, one of these algorithms extracts spike amplitude, is illustrated above, another algorithm extract spike widths. Neuronunits feature extraction algorithms were necessary to guide the evolution of the Genetic Algorithm.

**A paragraph to add somewhere inside the body of the poster:**

Importantly the entire software optimization program is featured inside a dedicated docker container. Such containers provide an environment that is reproducible independently of the host operating system and package management state. Previously the resolution of package dependency issues was assumed to be trivial, as may still be the case for simpler programs.

Increasingly products whose achievements rely on functional interfaces between other cutting-edge programs and portals accumulate a tedious amount of package management complexity, which is essentially entirely unrelated to the scientific questions at hand. When a model user or developer inherits an environment via a docker container they benefit from a series of predefined interventions, at the package management level, and thus they are relieved of an entirely administrative and scientifically uninformative burden.

The package management issues although essentially resolvable, are often also irreducibly complex, when two interfacing programs are undergoing development. In theory docker shifts the responsibility of a standard package configuration from the user back to the developer. In the long-term users of the *NeuronUnit* should be guaranteed an exceptionally standardized and maintained software environment. Utilization of Docker is an important statement, asserting that some of the burden model reproducibility may extend all the way down to the operating system level.

Probably redundant or interchangeable sentence:

