Balancing Generality and Accuracy with Portable, Customizable, Black-Box Performance Modeling

James Stevens

December 15, 2017

Acknowledgements

- Andreas Klöckner
- Matt Wala
- Kaushik Kulkarni

Goal: Model and Predict Performance

Why?

- *Algorithm Design*
  Provide programmer with insight into which aspects of workload contribute most to cost
- *Performance Optimization*
  Select fastest algorithm in configuration space
  - Runtime performance tuning (requires fast prediction)
- *Load Balancing*
  Provide accurate predictions of workload execution times, enabling better scheduling

Goal: Model and Predict Performance

What kind performance model do we want?

- *Analytical*
  Model should provide insight about what aspects of algorithm contribute to overall workload; no machine learning
- *Simple*
  Evaluation of model should be fast enough to enable runtime performance prediction
- *Platform Independent, Black Box*
  Model should be portable and should not reqire knowledge of (or manual input of) hardware stats

Simple Example - Model Mat-mul on GTX Titan X GPU

Predict execution time for square tiled matrix multiplication

- What's the simplest analytical model we can think of?

Simple Example - Model Mat-mul on GTX Titan X GPU

Predict execution time for square tiled matrix multiplication

- What's the simplest analytical model we can think of?
- $t = \text{loads} * p_{\text{loads}}$

Simple Example - Model Mat-mul on GTX Titan X GPU

Predict execution time for square tiled matrix multiplication

- What's the simplest analytical model we can think of?
- $t = \text{loads} * p_{\text{loads}}$
  - How can we determine p_{loads} if hardware is a black box?

Simple Example - Model Mat-mul on GTX Titan X GPU

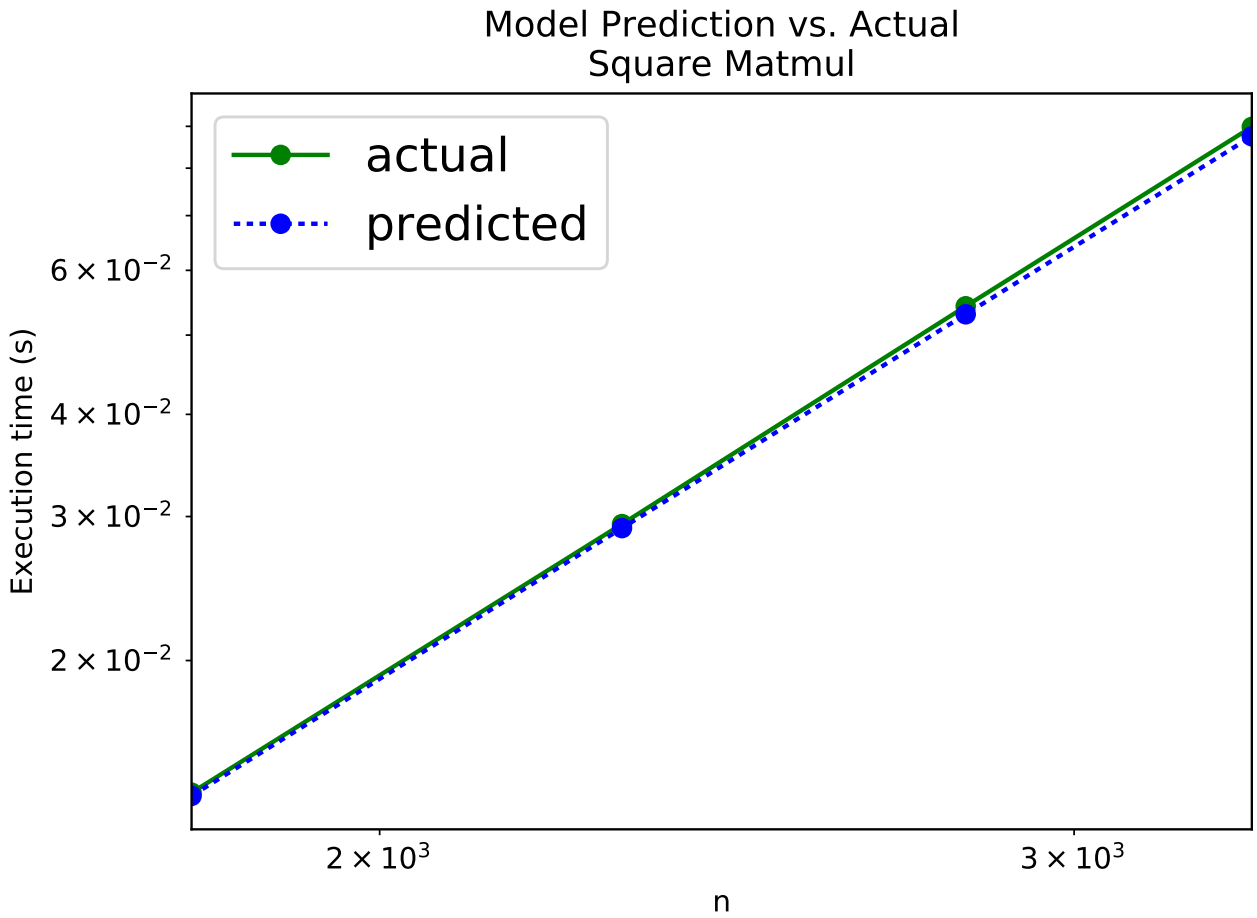
Predict execution time for square tiled matrix multiplication

- What's the simplest analytical model we can think of?
- $t = \text{loads} * p_{\text{loads}}$
  - How can we determine p_{loads} if hardware is a black box?
- Run a measurement computation to calibrate model

Simple Example - Model Mat-mul on GTX Titan X GPU

Quick demo.

Simple Example - Model Mat-mul on GTX Titan X GPU



Motivation

What if we want to accurately predict multiple matmul algorithms? Other computations?

- Need more complicated, general model
- Additional *features* (kernel properties)
- Run additional measurement kernels on GPU
- May need to sacrifice some accuracy for generality

Motivation

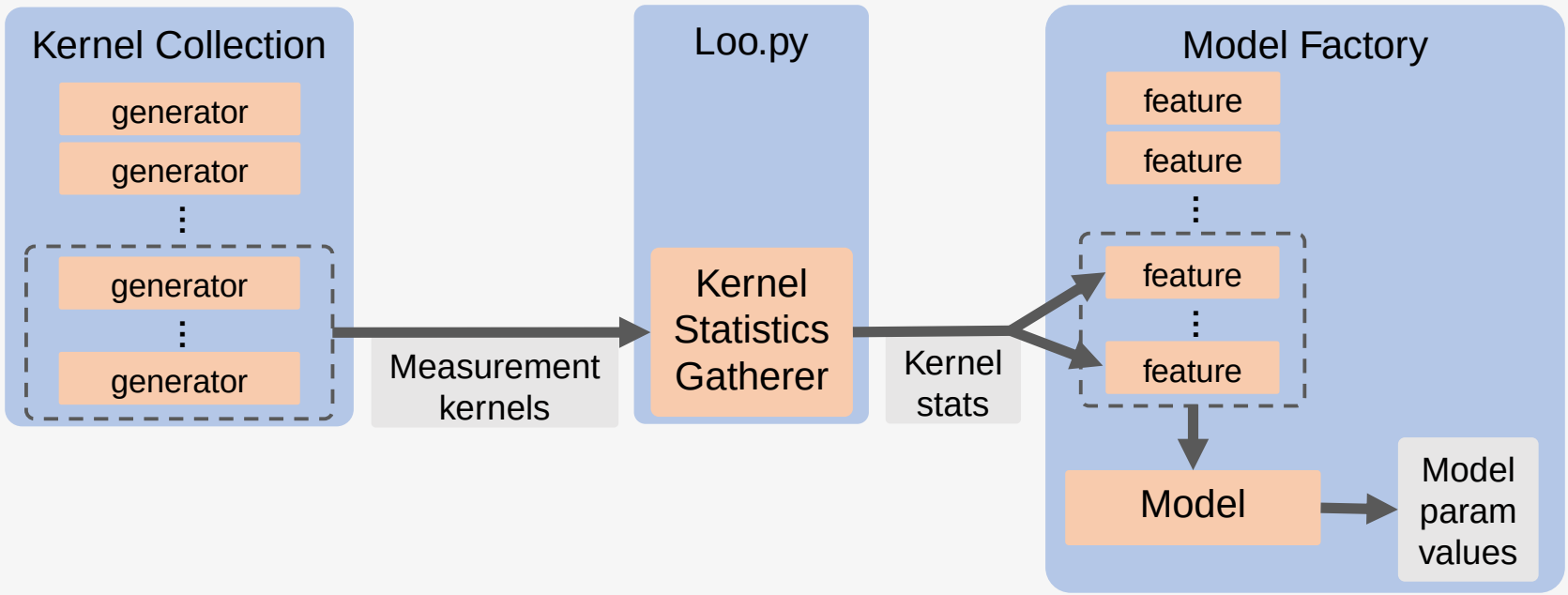
What if we want to accurately predict multiple matmul algorithms? Other computations?

- Need more complicated, general model
- Additional *features* (kernel properties)
- Run additional measurement kernels on GPU
- May need to sacrifice some accuracy for generality
- **Different applications may have different accuracy/generality requirements**

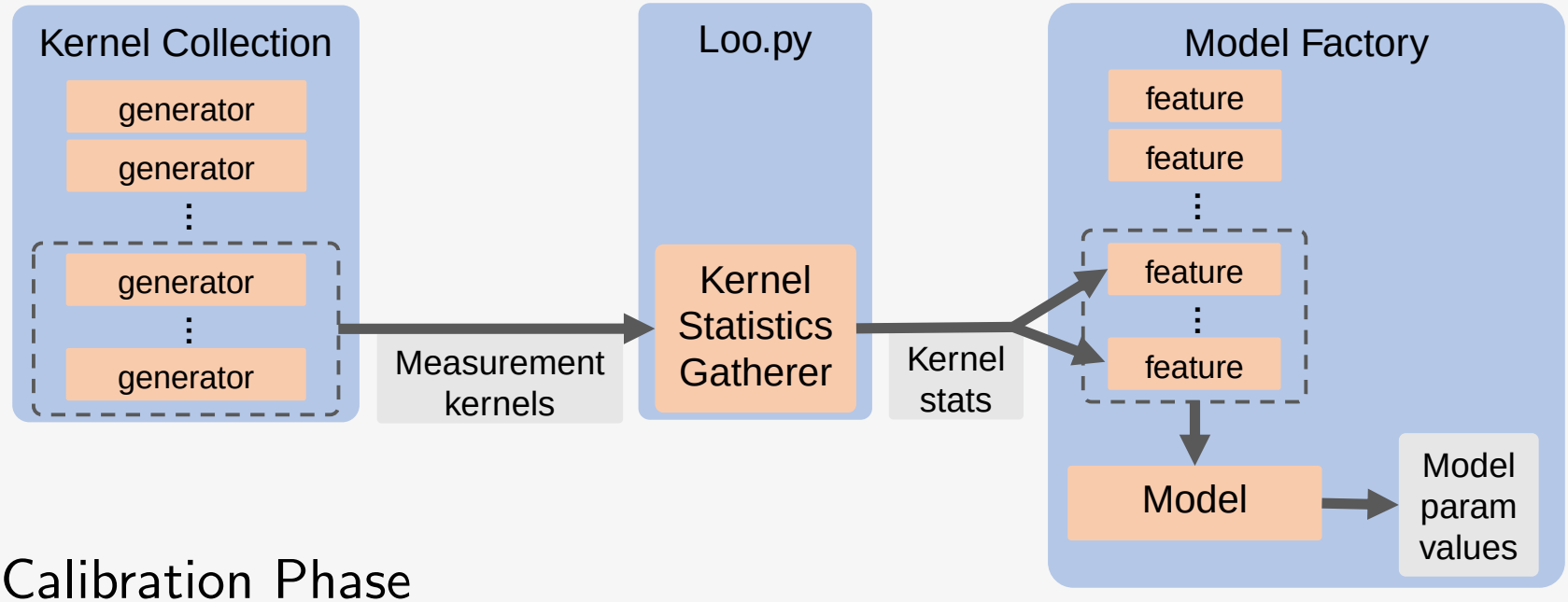
Motivation

Solution: Let the developer build a model that meets their needs

Software Overview



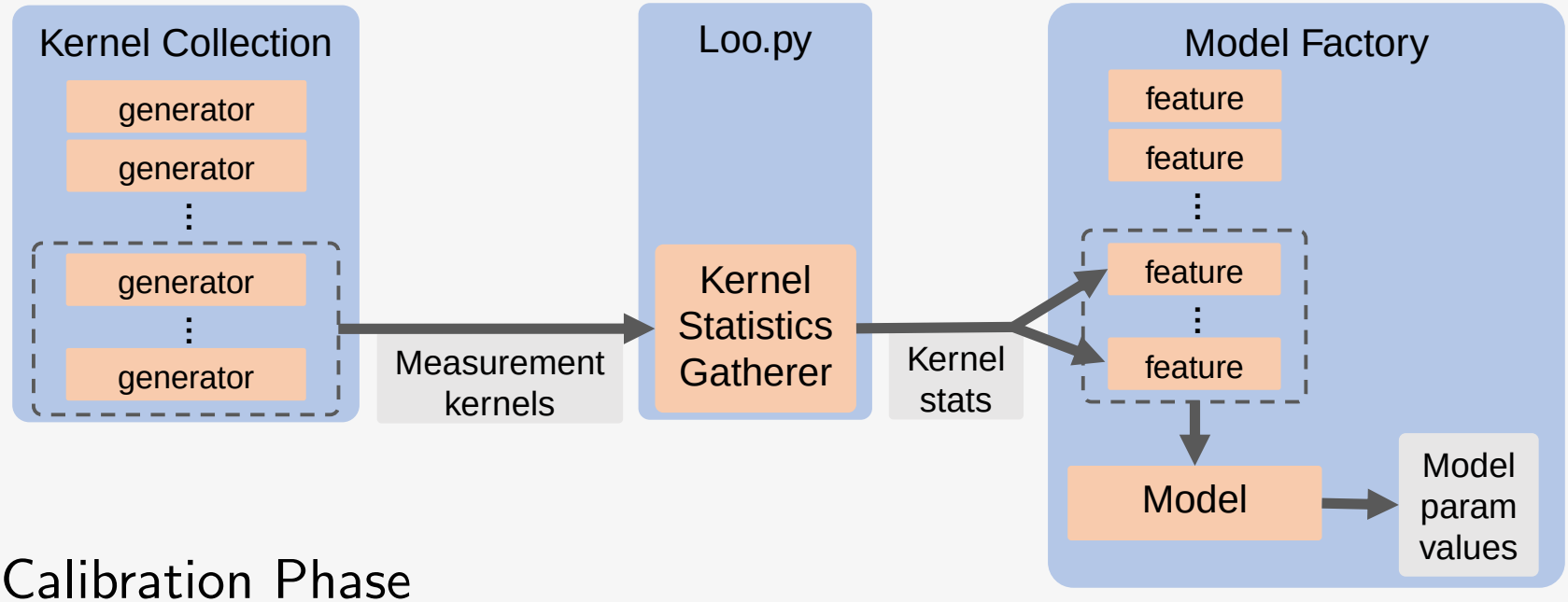
Software Overview



Calibration Phase

1. Create model expression using available kernel features
   - May create custom features

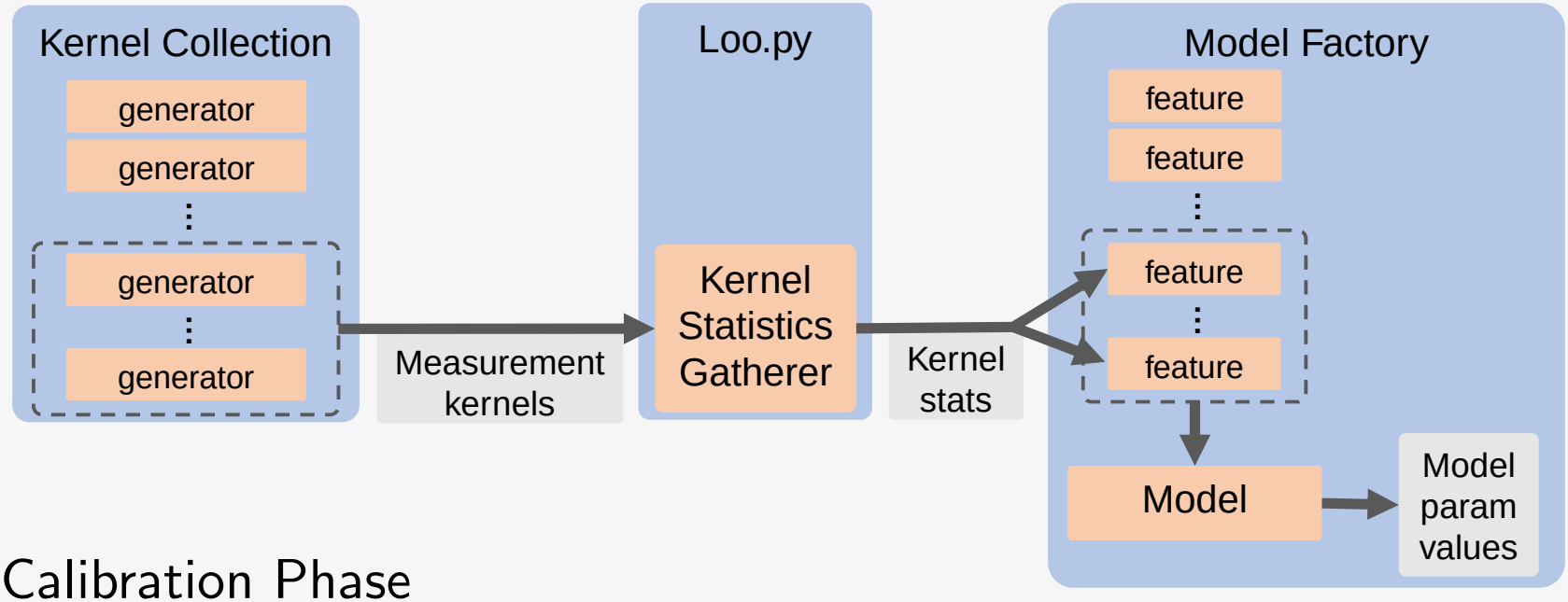
Software Overview



Calibration Phase

1. Create model expression using available kernel features
   - May create custom features
2. Filter kernel generators to produce measurement kernel set
   - May create custom measurement kernels

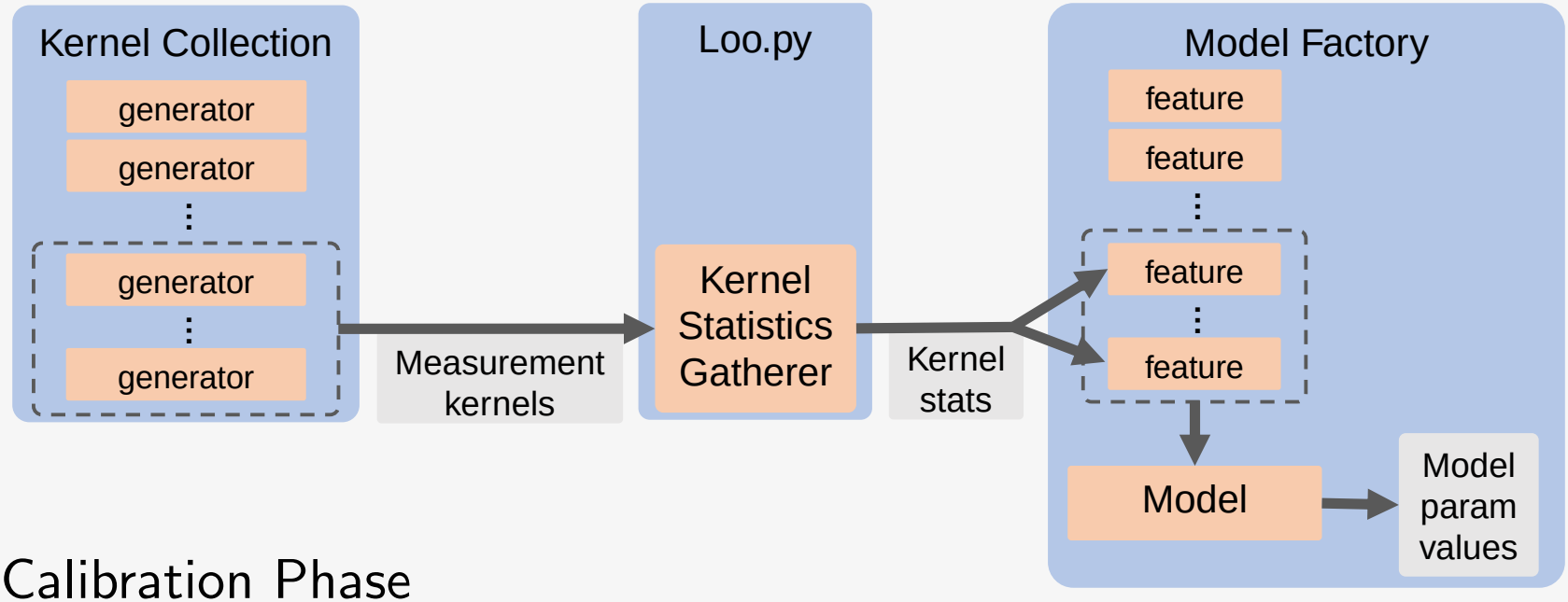
Software Overview



Calibration Phase

1. Create model expression using available kernel features
   - May create custom features
2. Filter kernel generators to produce measurement kernel set
   - May create custom measurement kernels
3. Compute features for each kernel using Loopy stats gatherer

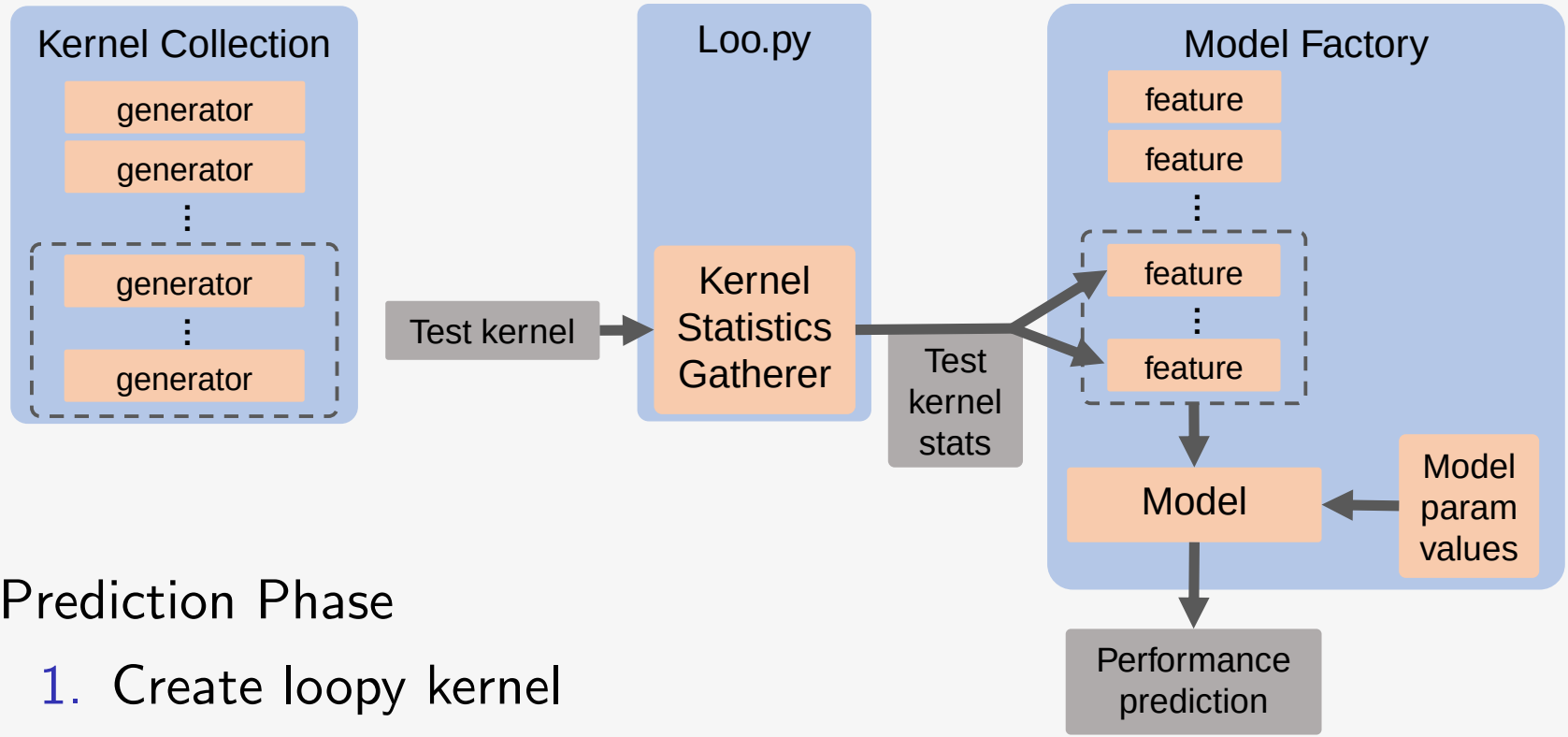
Software Overview



Calibration Phase

1. Create model expression using available kernel features
   - May create custom features
2. Filter kernel generators to produce measurement kernel set
   - May create custom measurement kernels
3. Compute features for each kernel using Loopy stats gatherer
4. Fit Model

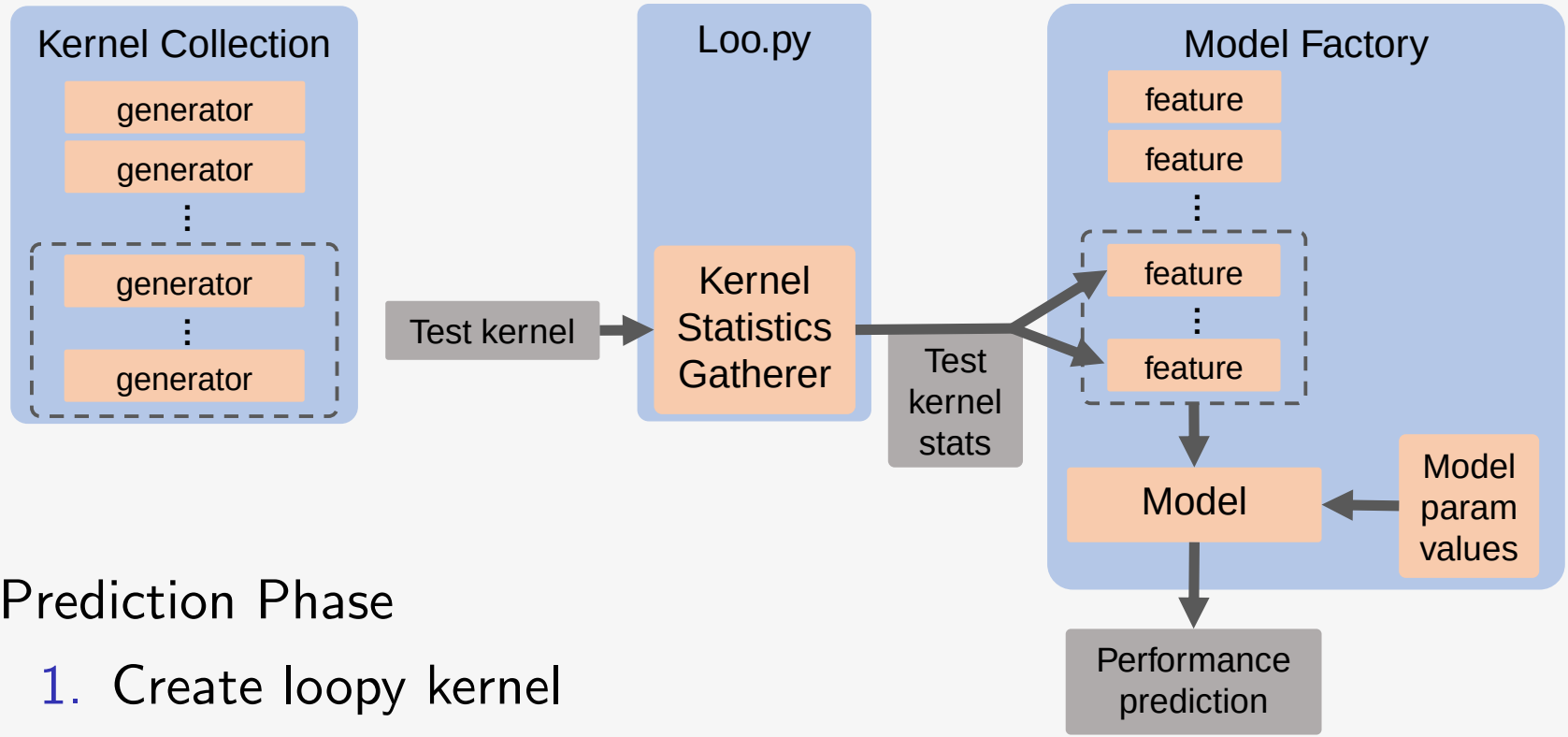
Software Overview



Prediction Phase

1. Create loopy kernel

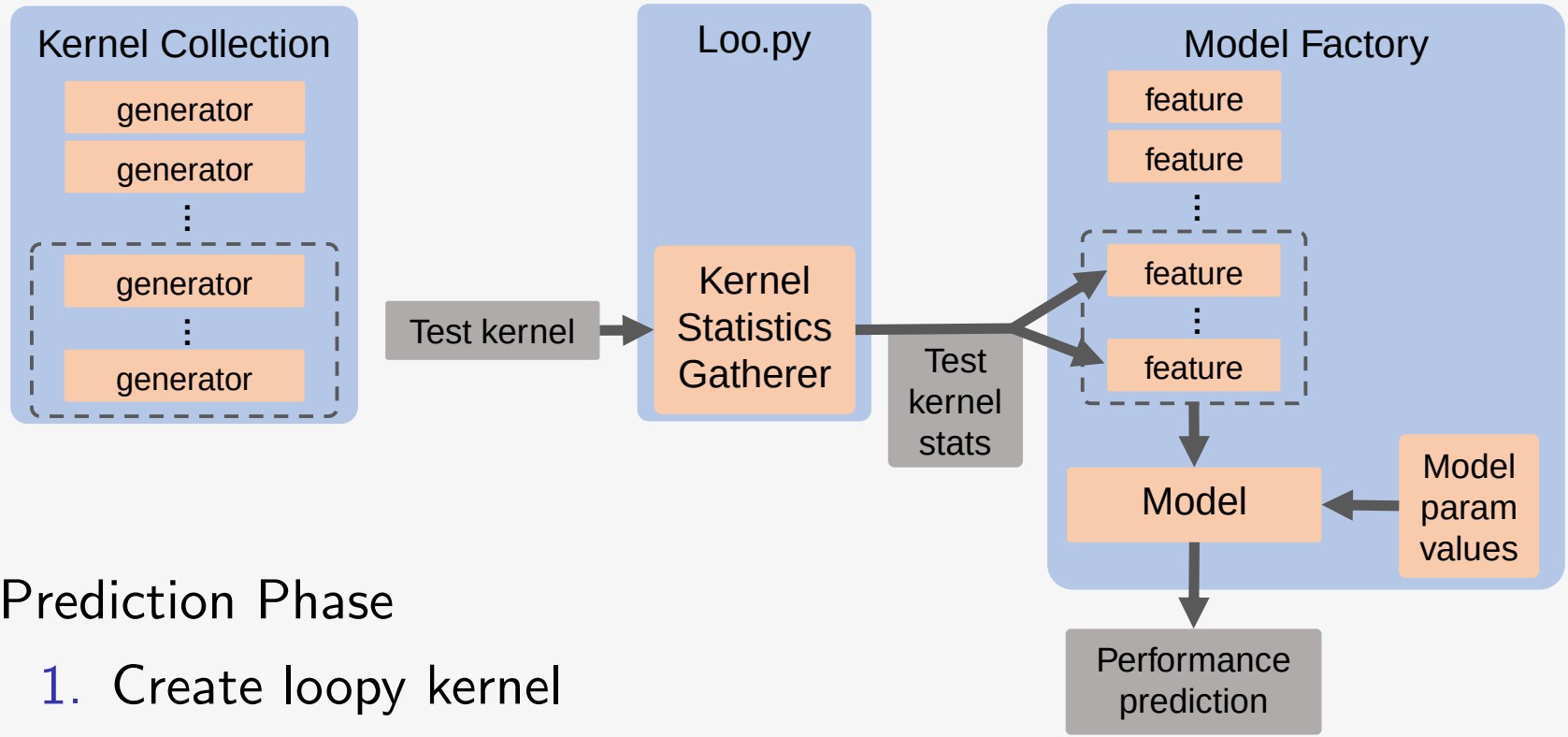
Software Overview



Prediction Phase

1. Create loopy kernel
2. Compute features for kernel using Loopy stats gatherer

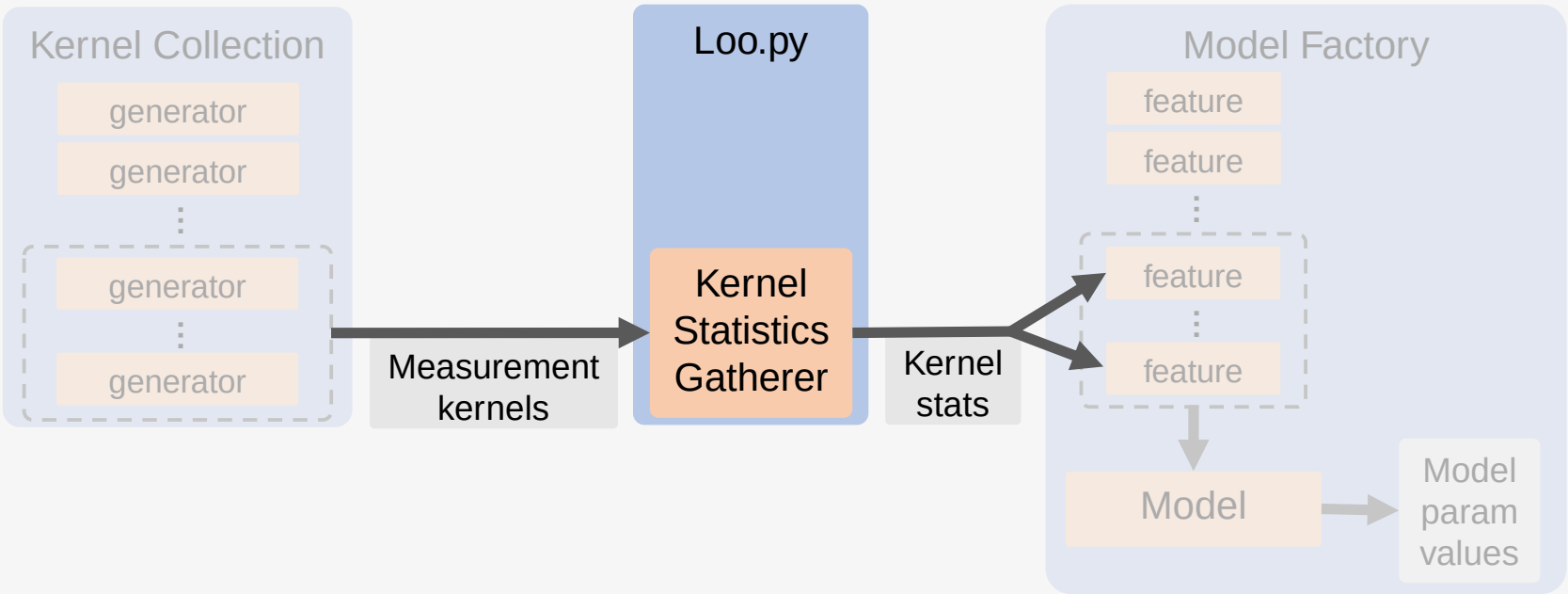
Software Overview



Prediction Phase

1. Create loopy kernel
2. Compute features for kernel using Loopy stats gatherer
3. Use model with parameter values from calibration phase to produce prediction

Counting Statistics with Loopy



Counting Statistics with Loopy

Kernel stats collected

- Memory Traffic
  - Categorize by data type, memory type, direction
  - Info on memory access pattern: "stride"
- (FL)OP/s: $+/-$, $*$, $\div$, $\wedge$
  - Categorize by data type, operation type
- Synchronization
  - Launch, local barrier, global barrier

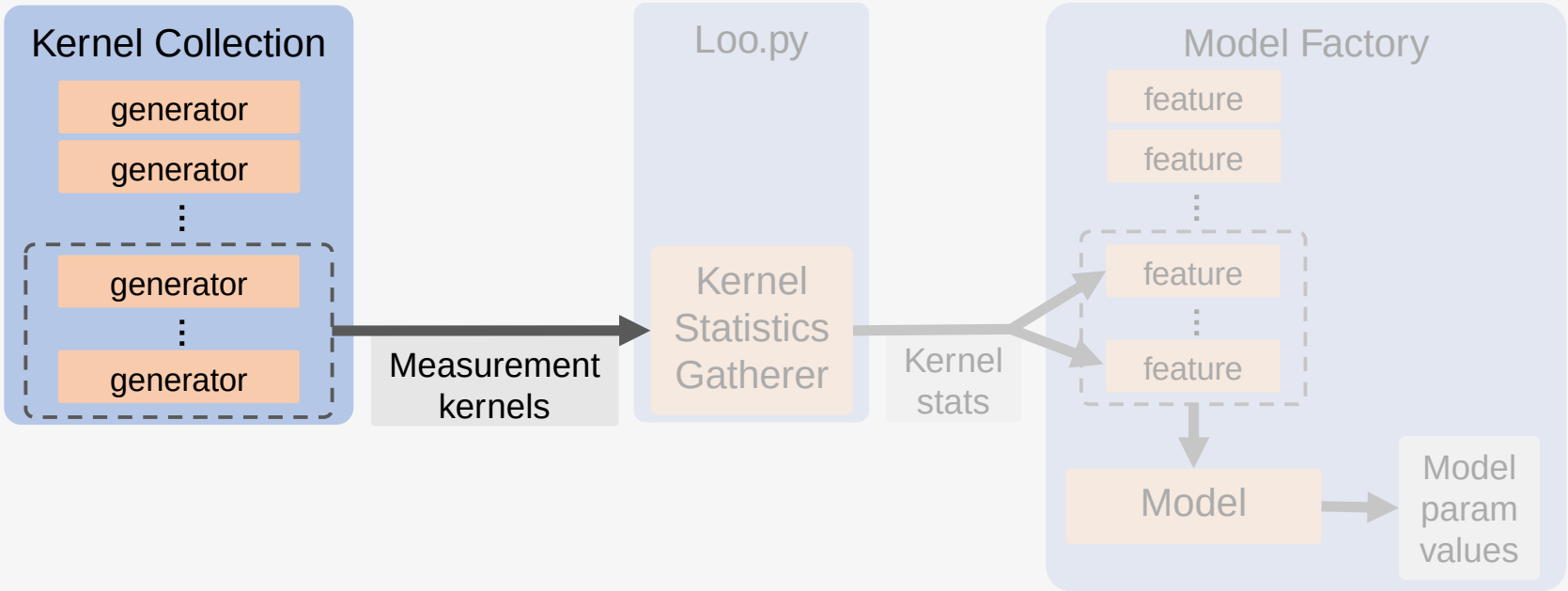
Counting Statistics with Loopy

```
knl = lp.make_kernel(
        "{[i,j]: 0<=i,j<n}",
        "a[i,j]=b[j,i]")
```

How do we count?

1. Recursively traverse **instruction expression tree** of a Loopy kernel, counting stats for single instruction
2. Determine how many times instruction executes
   - Barvinok counting library
     S. Verdoolaege et. al. Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions, Algorithmica, v.48 n.1, March 2007

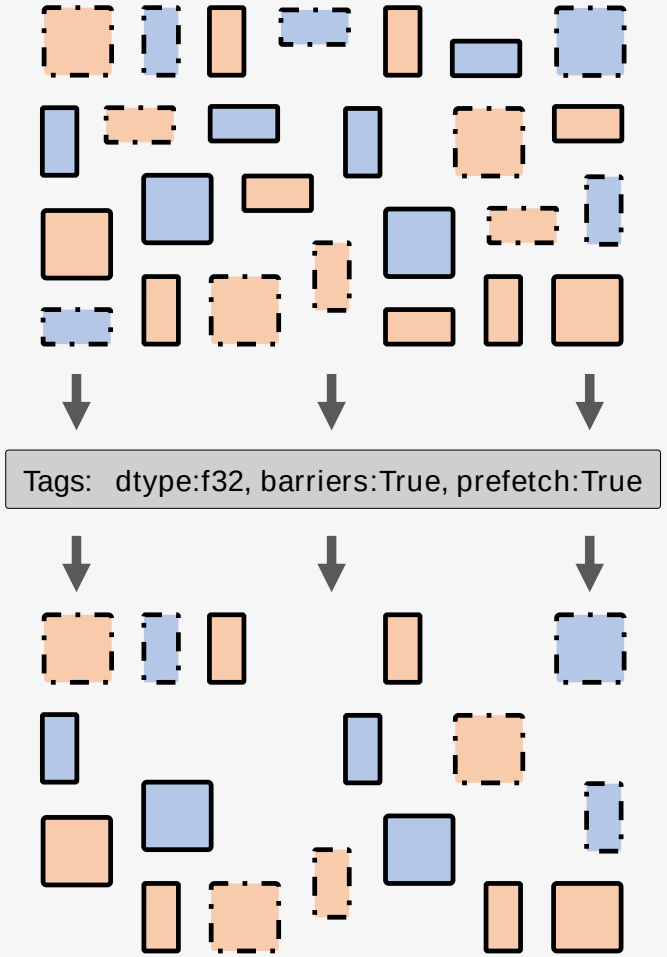
Generating Kernels with KernelCollection



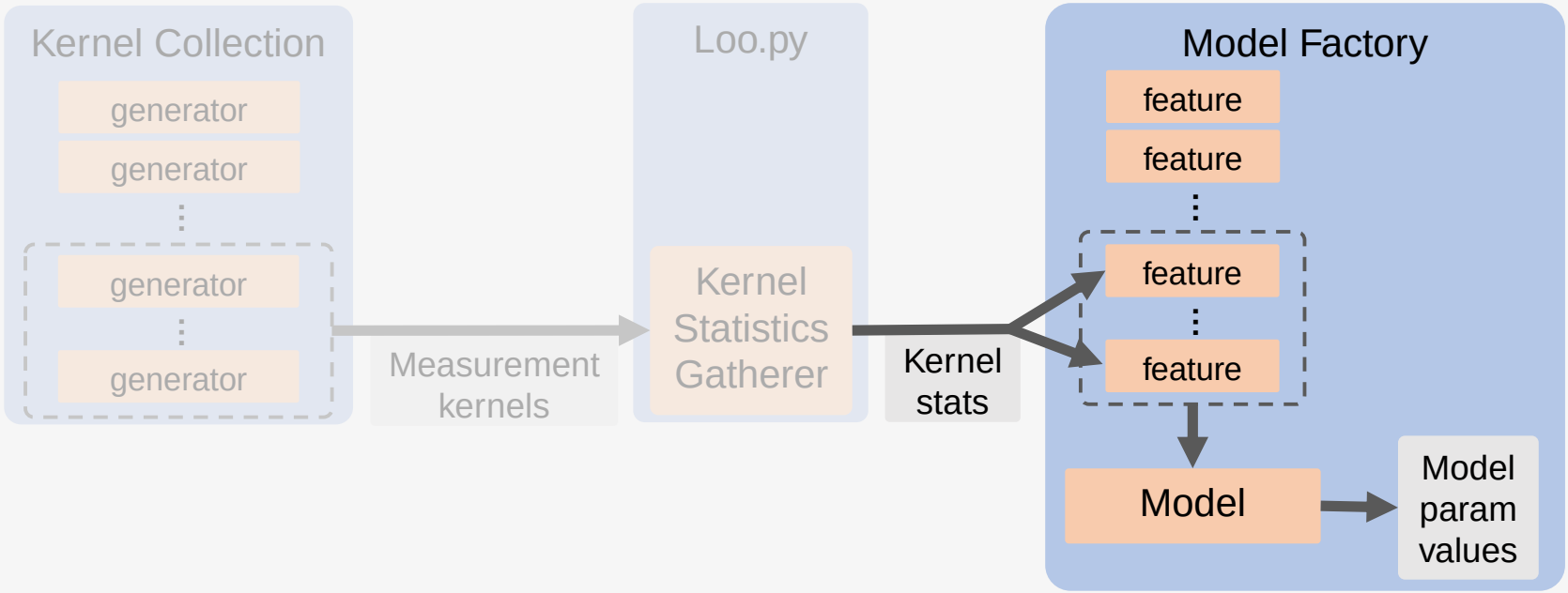
Generating Kernels with KernelCollection

Kernel generator functions produce measurement kernel set

- Use customizable tags to control which kernels will be produced
  - Generators are also customizable
- Example filtering:
  - Produce only kernels operating on float64 data
  - Produce only kernels that don't use local memory



Creating Model and Features with ModelFactory



Creating Model and Features with ModelFactory

Feature

- Object with an eval(knl) function that returns a numeric value characterizing the kernel
  - Number of stride-1 float32 global memory loads
  - Number of thread groups
- May create custom features

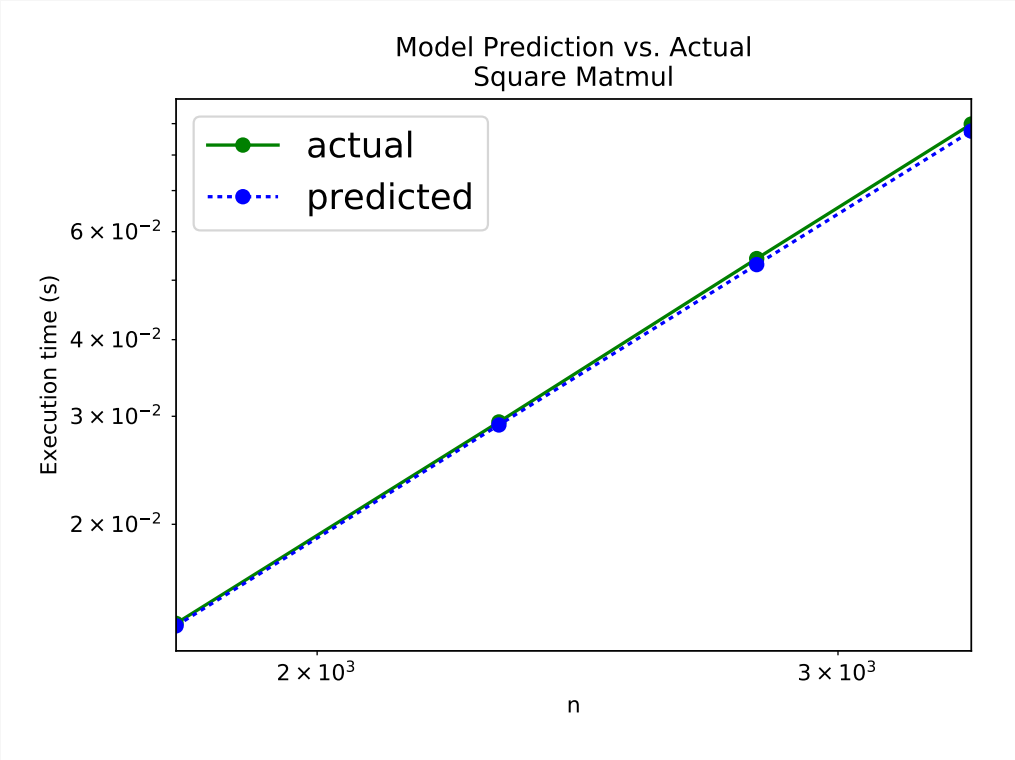
Creating Model and Features with ModelFactory

Model

- Mathematical expression relating input features, parameters to output feature
- $t = \text{loads} * p_{\text{loads}}$
- Gather features for all measurement kernels, then fit model using nonlinear least squares to solve for model parameters

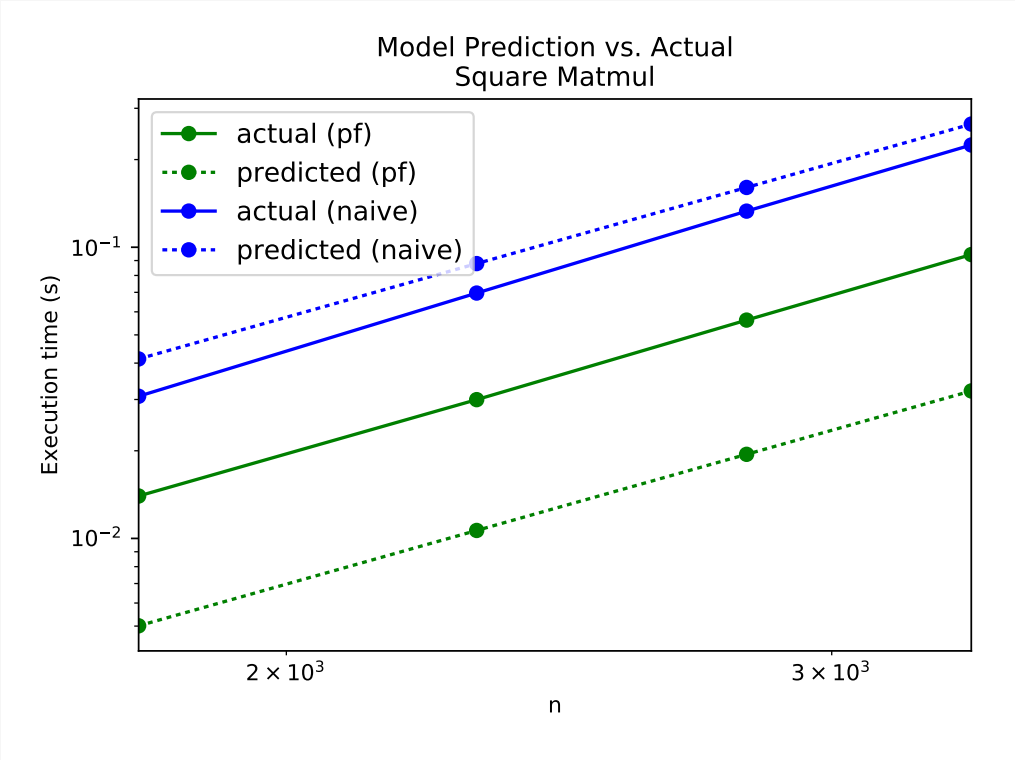
Matrix Multiplication Model - Version 1

- $t = \text{loads} * p_{\text{loads}}$
- Tiled matmul with prefetching



Matrix Multiplication Model - Version 1

- $t = \text{loads} * p_{\text{loads}}$
- Naive matmul **and** Tiled matmul with prefetching
- What happened? Different mem access patterns, different effective mem access costs, different types of memory used

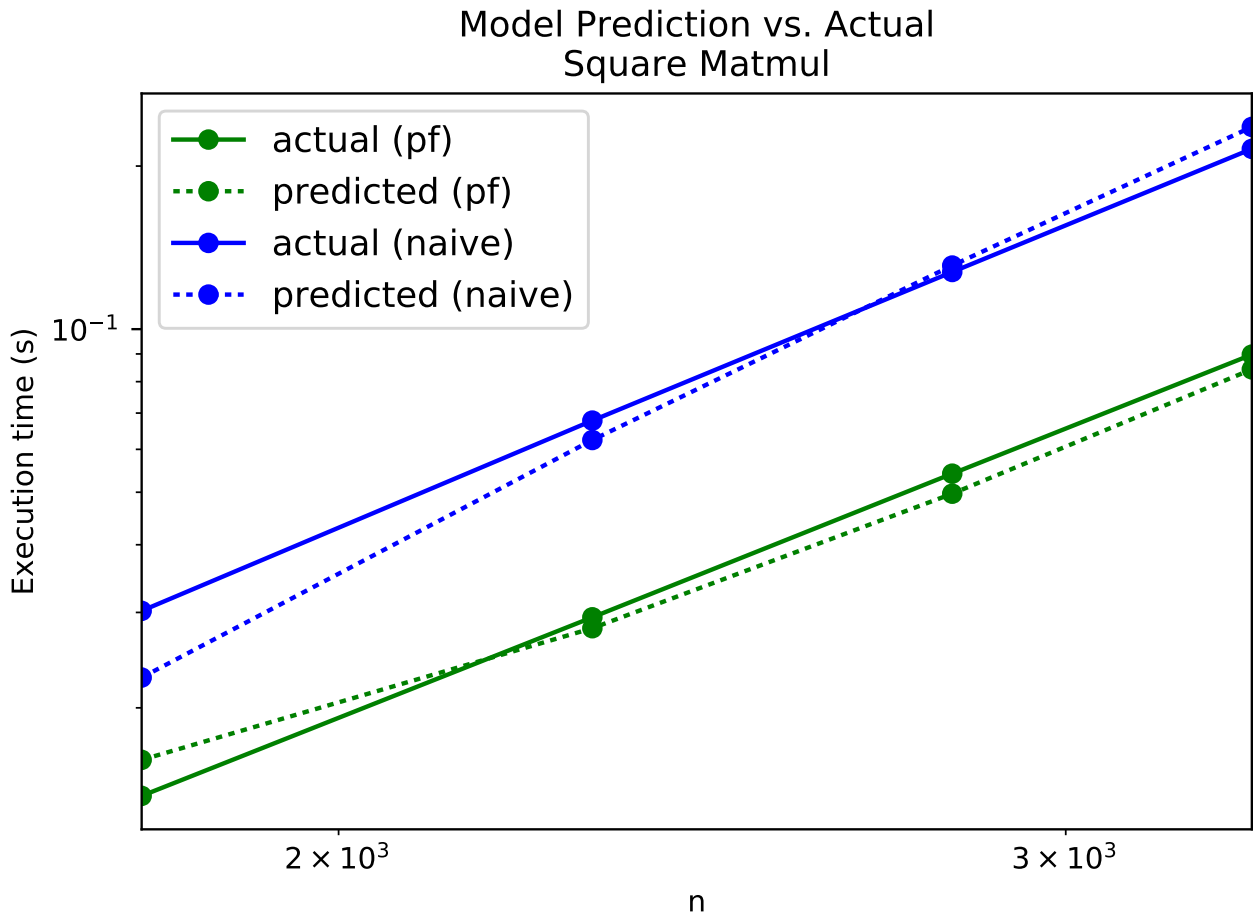


Matrix Multiplication Model - Version 2

$$
\begin{aligned}
t = &\text{gloads-s1-b4} * p_{\text{gl-s1-b4}} \\
&+ \text{gloads-s1-b2} * p_{\text{gl-s1-b2}} \\
&+ \text{gstores-s1-b4} * p_{\text{gs-s1-b4}} \\
&+ \text{lloads} * p_{\text{ll}} \\
&+ \text{lstores} * p_{\text{ls}} \\
&+ \text{mult-ops} * p_{\text{mult}} \\
&+ \text{add-ops} * p_{\text{add}} \\
&+ \text{thread-groups} * p_{\text{g-overhead}} \\
&+ \text{kernel-launches} * p_{\text{k-overhead}}
\end{aligned}
$$

- Need additional measurement kernels
  - Memory traffic, add/mult ops, local mem access, empty kernel

Matrix Multiplication Model - Version 2



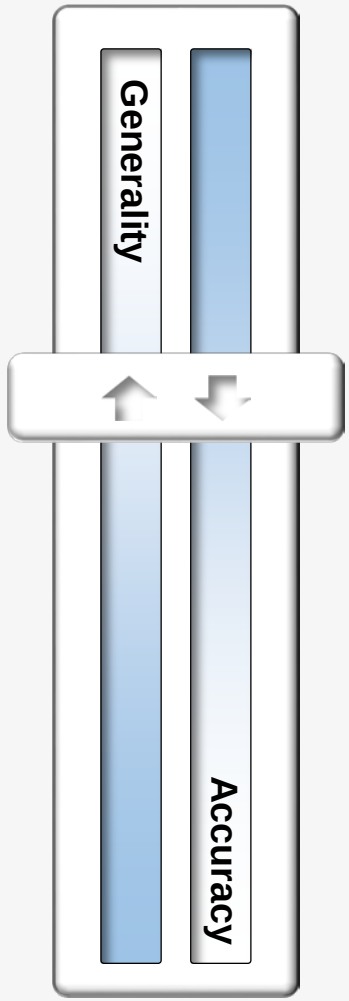
Modeling Nonlinear Relationships

- GPUs overlap local and global operations
- We can model this!
- Let V, G, L be overhead, global, and local costs
- Want: $t = V + max(G, L)$, but max() is not differentiable
- Instead:

$$t = V + G * (tanh(G - L) + 1) * 0.5 + L * (tanh(L - G) + 1) * 0.5$$

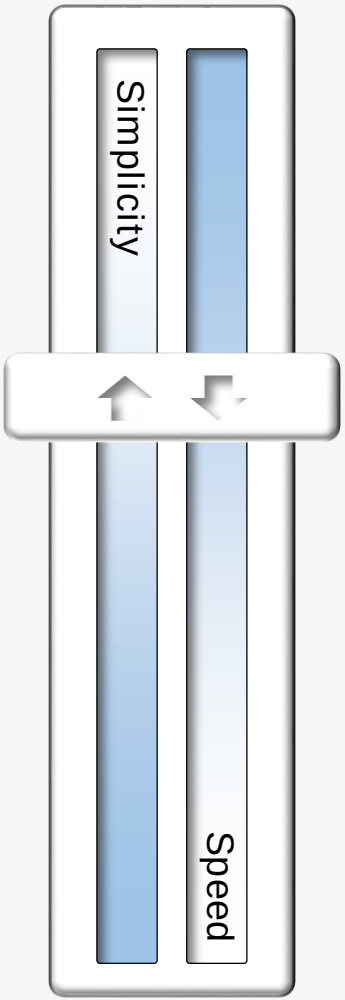
Accuracy vs. Generality

- Increasing model generality decreases accuracy
- We provide an accuracy-vs-generality knob for controlling this tradeoff



Simplicity vs. Speed

- Increasing model complexity decreases speed
- Most models built with this software will be much faster to evaluate than complicated non-black-box options, but some will be faster than others
- We allow the developer to make the model as simple as necessary to achieve the desired evaluation time



End.