Applied Urban Modeling
November 30, 2020

Using Machine Learning and GPU Processing to Build Faster and More Accurate Integrated Models

Paul Waddell

Professor, City and Regional Planning
University of California, Berkeley
Director, Urban Analytics Lab
President, UrbanSim Inc.

AGENDA

01 **ML Hedonic Models to Bootstrap Price Predictions**

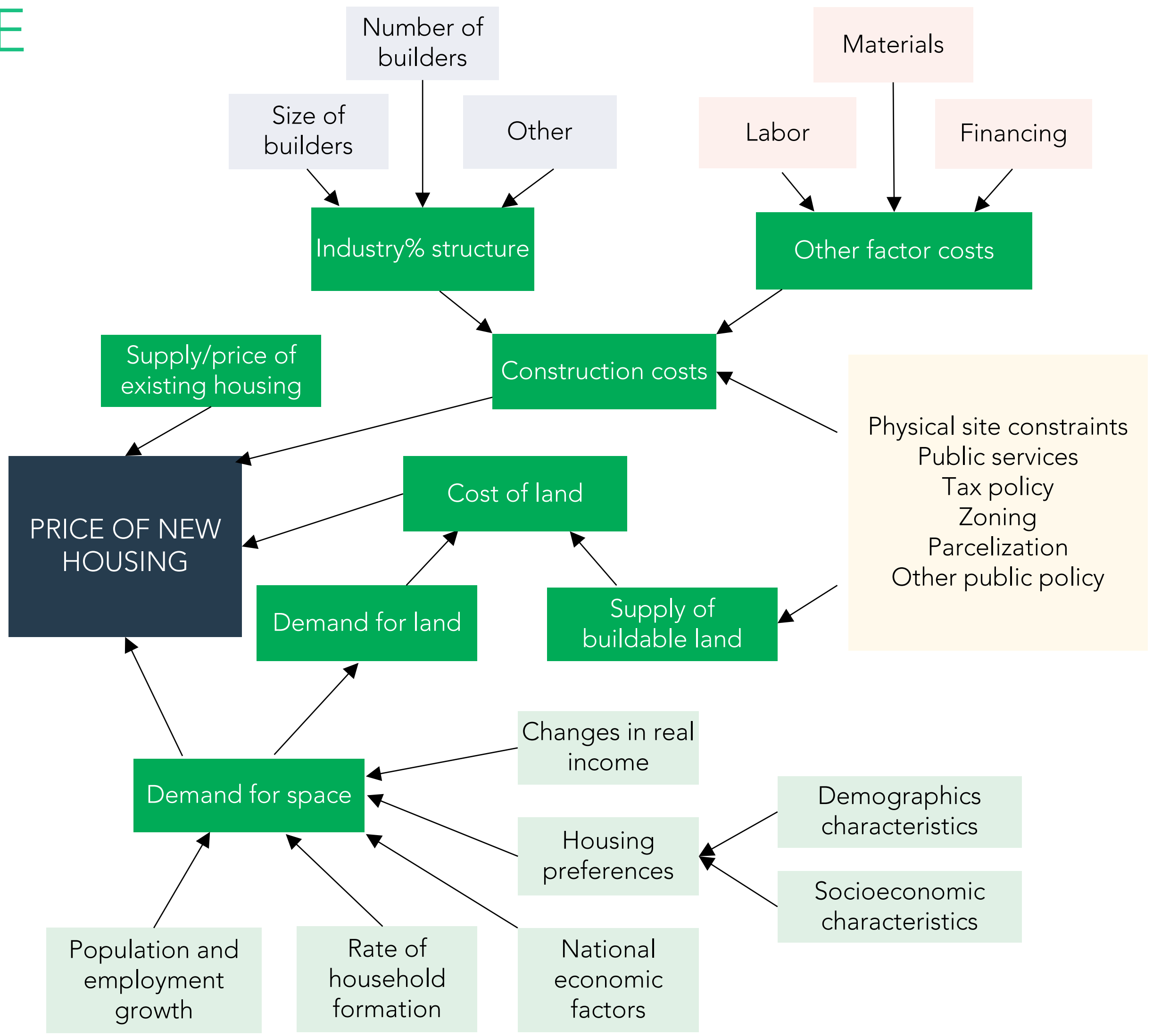
02 Differentiable Models for Calibration and Price Equilibration

03 GPU based Traffic Microsimulation

PLANNING INTERVENTIONS OPERATE ON AND WITHIN COMPLEX SYSTEMS

Simulation models of these systems tend to be complex and structural in nature, to support counter-factual analysis.

Interpretation and causal reasoning is prized over predictive accuracy.



STATISTICAL MODELS VS MACHINE LEARNING

Statistical models for regression, discrete choice

- Dominant for transport & land use modeling
- Causal interpretation (with caveats)
- Favored for counter-factual analysis

Machine Learning models for regression, classification

- Excel at predictive accuracy
- Handle big data and nonlinear relationships
- Methods to avoid overfitting
- Lower interpretability

USE CASE FOR ML IN URBAN SIMULATION MODELS

Hedonic Regression of Rents

- 360K Craigslist listings for San Francisco Bay Area from 2017-18
- Parsimonious covariates, focused on accessibility and spatial context

Rent Predictions used to Bootstrap Structural Model

- Discrete choice of housing demand by tenure (within year) depends on rents
- Fixed housing supply (within year)
- Bootstrap prices and rents adjust to clear the market (within year)
- Pro forma model of housing supply (between years) depends on rents

Predictive Accuracy of Bootstrap Rents Outweighs Interpretability

DEPENDENT VARIABLE: CRAIGSLIST RENTS PER SQUARE FOOT PREDICTORS: NETWORK AGGREGATION QUERIES

	count	mean	std	min	25%	50%	75%	max
rent_sqft	363010.0	3.0	1.0	0.0	2.0	3.0	4.0	11.0
res_sqft_per_unit	363010.0	994.0	430.0	212.0	710.0	904.0	1150.0	3600.0
units_500_walk	363010.0	664.0	662.0	0.0	193.0	437.0	876.0	2317.0
sqft_unit_500_walk	363010.0	1455.0	712.0	0.0	1059.0	1436.0	1803.0	3699.0
rich_500_walk	363010.0	133.0	148.0	0.0	27.0	81.0	166.0	528.0
singles_500_walk	363010.0	201.0	254.0	0.0	35.0	101.0	228.0	868.0
elderly_hh_500_walk	363010.0	92.0	102.0	0.0	21.0	56.0	117.0	363.0
children_500_walk	363010.0	226.0	189.0	0.0	79.0	186.0	327.0	755.0
jobs_500_walk	363010.0	759.0	1295.0	0.0	43.0	220.0	748.0	5247.0
jobs_1500_walk	363010.0	6589.0	8770.0	0.0	1206.0	3110.0	7220.0	32501.0
jobs_10000	363010.0	165285.0	117970.0	0.0	74380.0	127551.0	236962.0	412326.0
jobs_25000	363010.0	498022.0	229898.0	37.0	322181.0	584284.0	696465.0	787748.0
pop_10000	363010.0	333207.0	191209.0	0.0	183445.0	300216.0	459446.0	763247.0
pop_black_10000	363010.0	14010.0	18451.0	0.0	2709.0	5754.0	20794.0	90219.0
pop_hisp_10000	363010.0	57468.0	42489.0	0.0	27776.0	45772.0	81072.0	201053.0
pop_asian_10000	363010.0	106511.0	77819.0	0.0	37199.0	93097.0	175019.0	282688.0

Note: variables are measured as network aggregations within 0.5, 1.5, 10 or 25 KM with Pandana
Data are for San Francisco Bay Area

THE MODEL

```
m = OLSRegressionStep()
m.tables = ['rentals', 'nodessmall_vars', 'nodeswalk_vars']
m.filters = ['rent_sqft < 10']
m.model_expression = 'np.log1p(rent_sqft) ~ +
    np.log(res_sqft_per_unit) + \
    np.log(units_500_walk+1) + np.log(sqft_unit_500_walk+2)  + \
    np.log(rich_500_walk + 1) + np.log(singles_500_walk + 1) + \
            np.log(elderly_hh_500_walk + 1) + \
    np.log(children_500_walk + 1) + \
    np.log(jobs_500_walk + 1) + np.log(jobs_1500_walk+1) + \
    np.log(jobs_10000+1) + np.log(jobs_25000 + 1) + \
    np.log(pop_10000+1) + np.log(pop_black_10000+1) + \
    np.log(pop_hisp_10000+1) + \
    np.log(pop_asian_10000+1)  '
m.out_column = 'pred_rent_sqft'
m.out_transform = np.expm1
m.name = 'hedonic_rent_sqft'

m.fit()
m.run()
mm.register(m)
```

Note: The OLS model is created, specified, fit, run, and registered (saved) using a new UrbanSim template library

OLS RESULTS

OLS Regression Results

Dep. Variable:	np.log1p(rent_sqft)	R-squared:	0.630
Model:	OLS	Adj. R-squared:	0.630
Method:	Least Squares	F-statistic:	2.745e+04
Date:	Tue, 31 Jul 2018	Prob (F-statistic):	0.00
Time:	14:56:02	Log-Likelihood:	1.1327e+05
No. Observations:	242063	AIC:	-2.265e+05
Df Residuals:	242047	BIC:	-2.263e+05
Df Model:	15		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.0244	0.013	155.472	0.000	1.999	2.050
np.log(res_sqft_per_unit)	-0.3260	0.001	-399.046	0.000	-0.328	-0.324
np.log(units_500_walk + 1)	-0.0154	0.001	-13.758	0.000	-0.018	-0.013
np.log(sqft_unit_500_walk + 2)	-0.0052	0.000	-14.193	0.000	-0.006	-0.004
np.log(rich_500_walk + 1)	0.0602	0.000	121.972	0.000	0.059	0.061
np.log(singles_500_walk + 1)	0.0141	0.001	19.289	0.000	0.013	0.016
np.log(elderly_hh_500_walk + 1)	0.0096	0.001	17.867	0.000	0.009	0.011
np.log(children_500_walk + 1)	-0.0510	0.001	-83.843	0.000	-0.052	-0.050
np.log(jobs_500_walk + 1)	0.0106	0.000	42.812	0.000	0.010	0.011
np.log(jobs_1500_walk + 1)	0.0016	0.000	5.778	0.000	0.001	0.002
np.log(jobs_10000 + 1)	0.0329	0.001	33.964	0.000	0.031	0.035
np.log(jobs_25000 + 1)	0.0916	0.001	102.946	0.000	0.090	0.093
np.log(pop_10000 + 1)	0.0673	0.002	34.157	0.000	0.063	0.071
np.log(pop_black_10000 + 1)	-0.0163	0.000	-44.192	0.000	-0.017	-0.016
np.log(pop_hisp_10000 + 1)	-0.0386	0.001	-50.321	0.000	-0.040	-0.037
np.log(pop_asian_10000 + 1)	-0.0272	0.001	-39.281	0.000	-0.029	-0.026

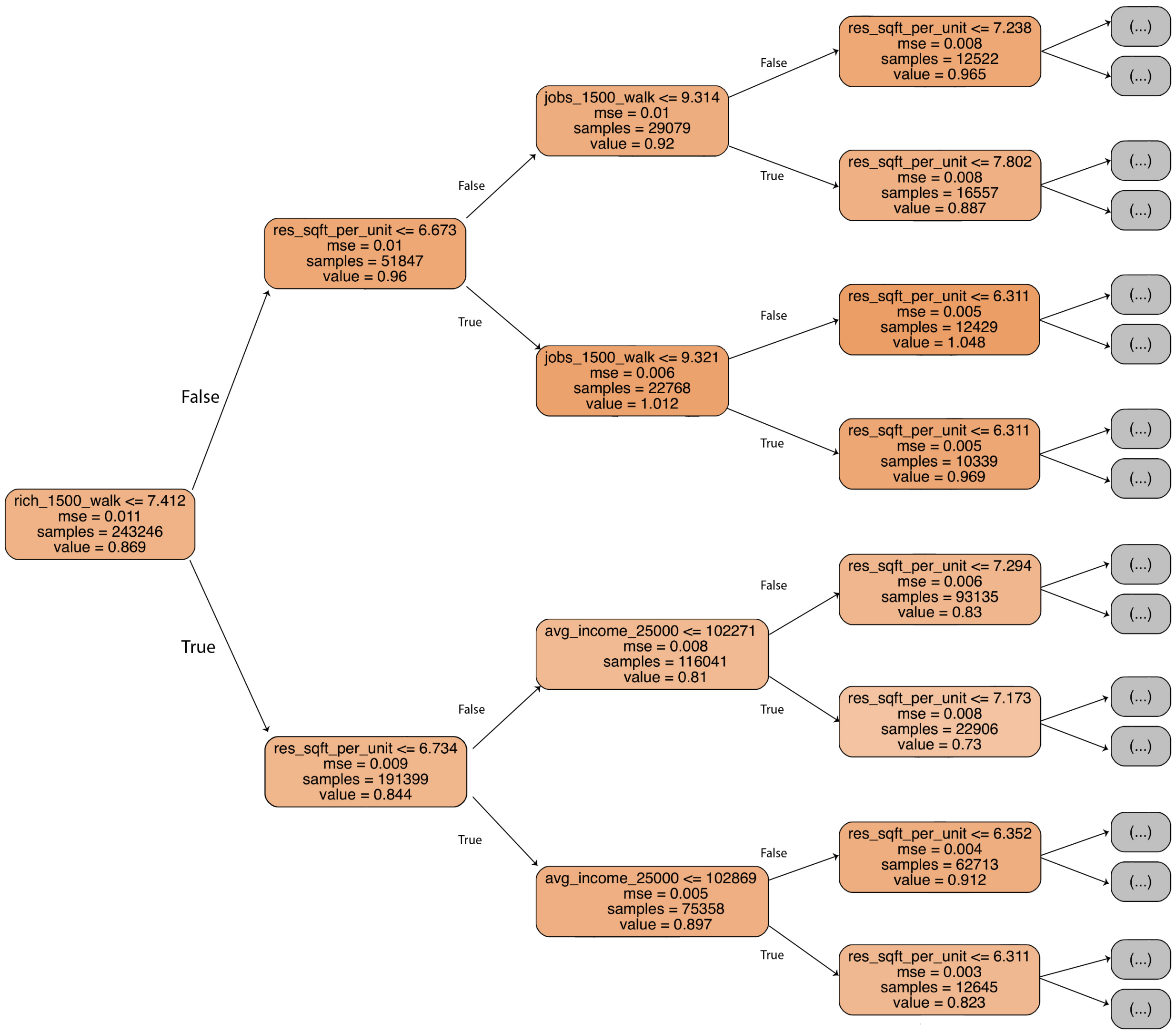
Omnibus:	21555.547	Durbin-Watson:	0.780
Prob(Omnibus):	0.000	Jarque-Bera (JB):	104004.384
Skew:	-0.310	Prob(JB):	0.00
Kurtosis:	6.151	Cond. No.	1.39e+03

Note: models were estimated using 2/3 of the data, retaining 1/3 to be used for validation

ELASTICITIES

Variable	Elasticity
Residential Sqft per unit	-0.33
Jobs within 25 kilometers	0.09
Population within 10 kilometers	0.07
Rich households within 1/2 kilometer	0.06
Children within 1/2 kilometer	-0.05

REGRESSION TREE

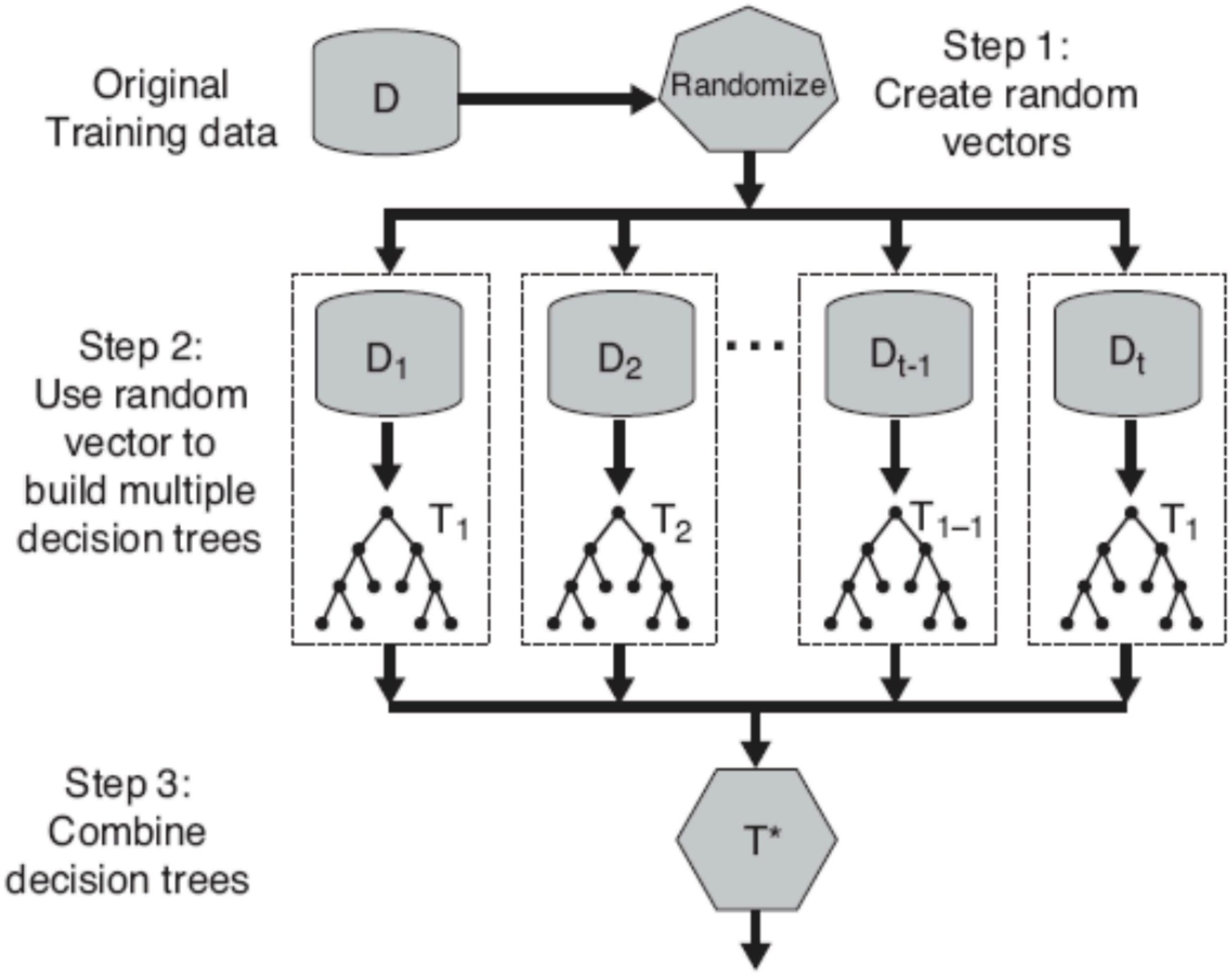


RANDOM FOREST

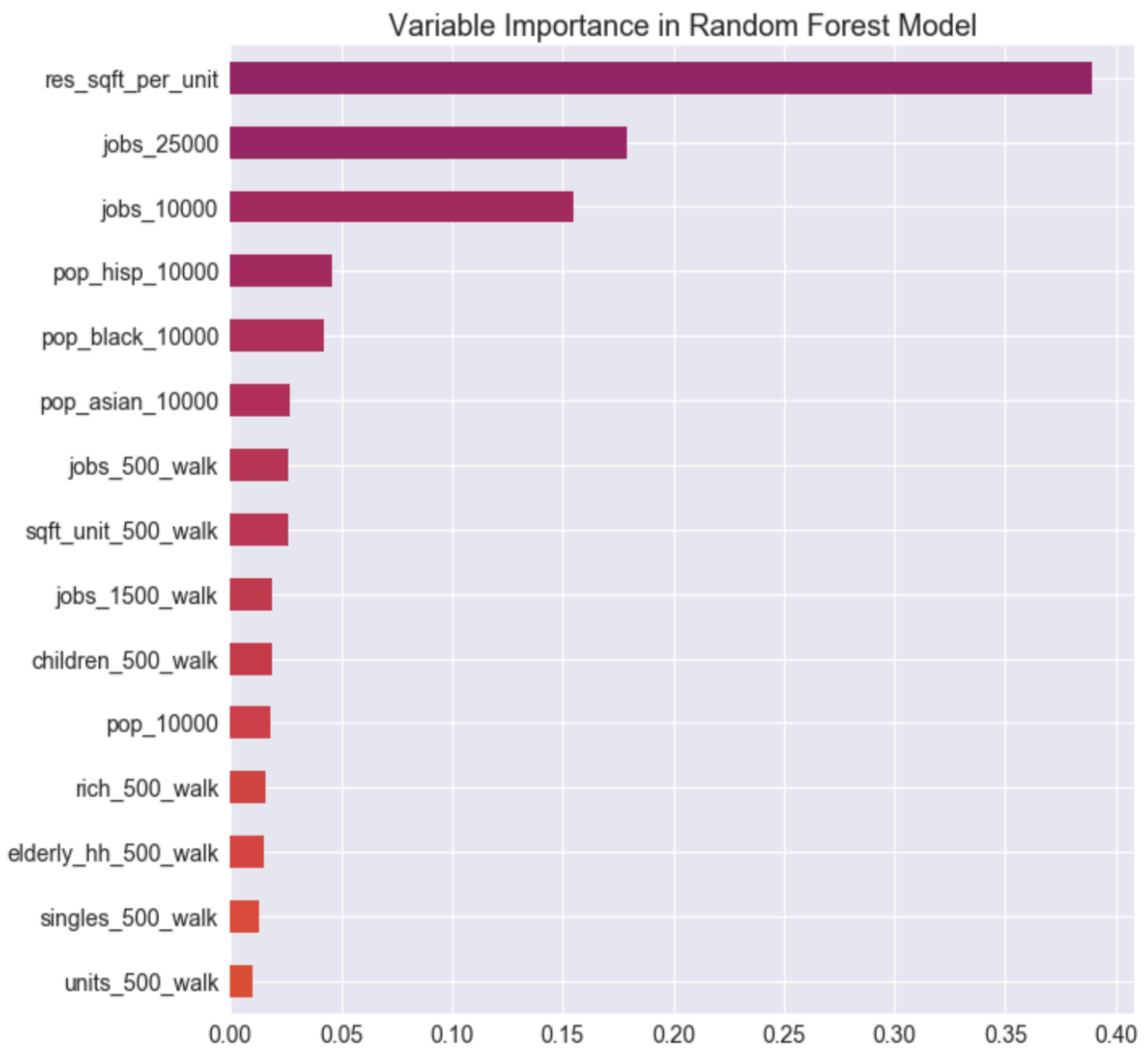
RF handles non-linear relationships between the dependent and independent variables

RF is invariant to scaling and translation

RF is robust to irrelevant or highly correlated variables

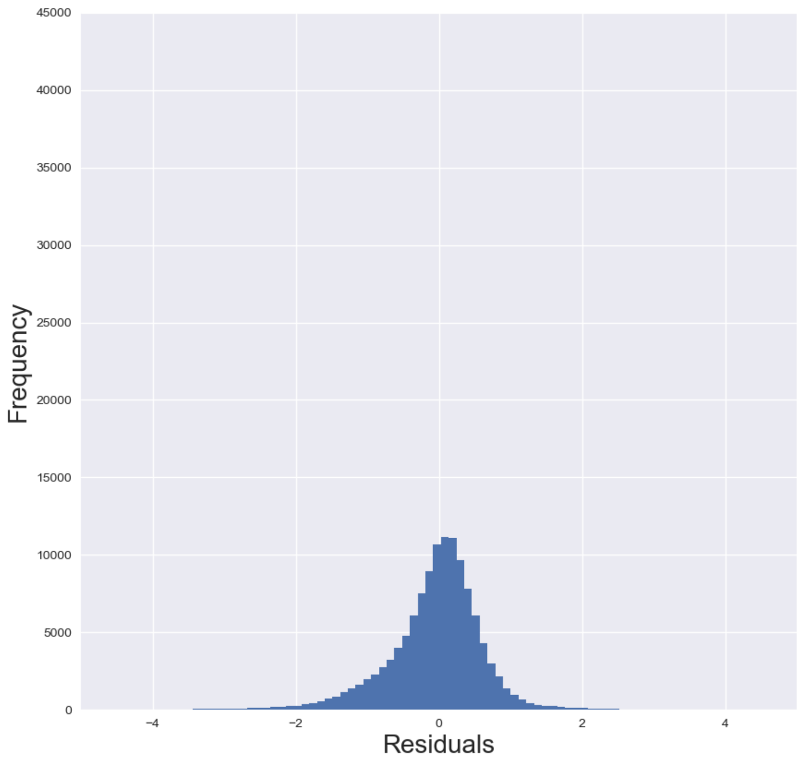


RANDOM FOREST

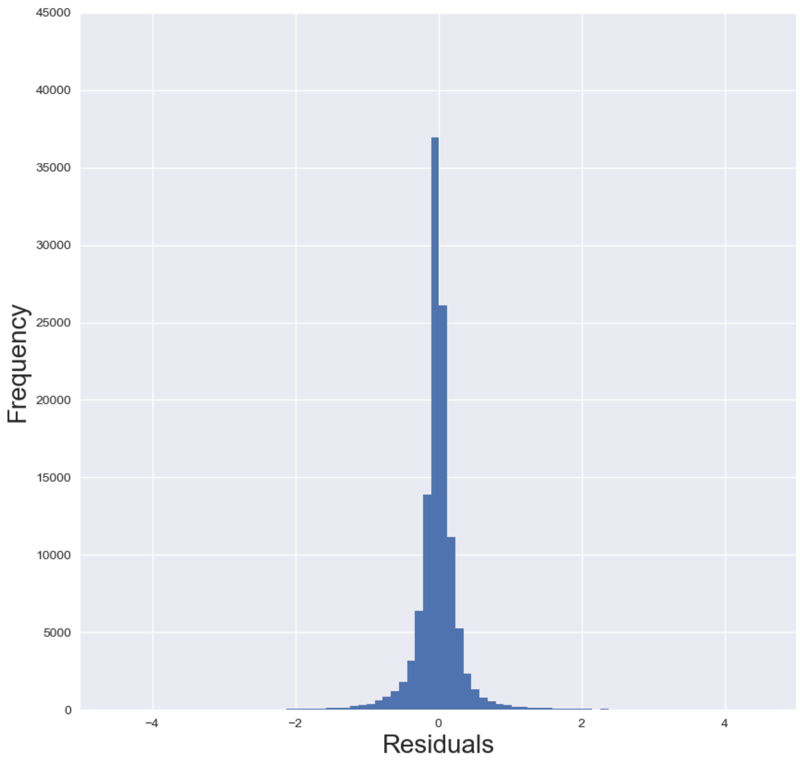


DISTRIBUTION OF RESIDUALS

OLS



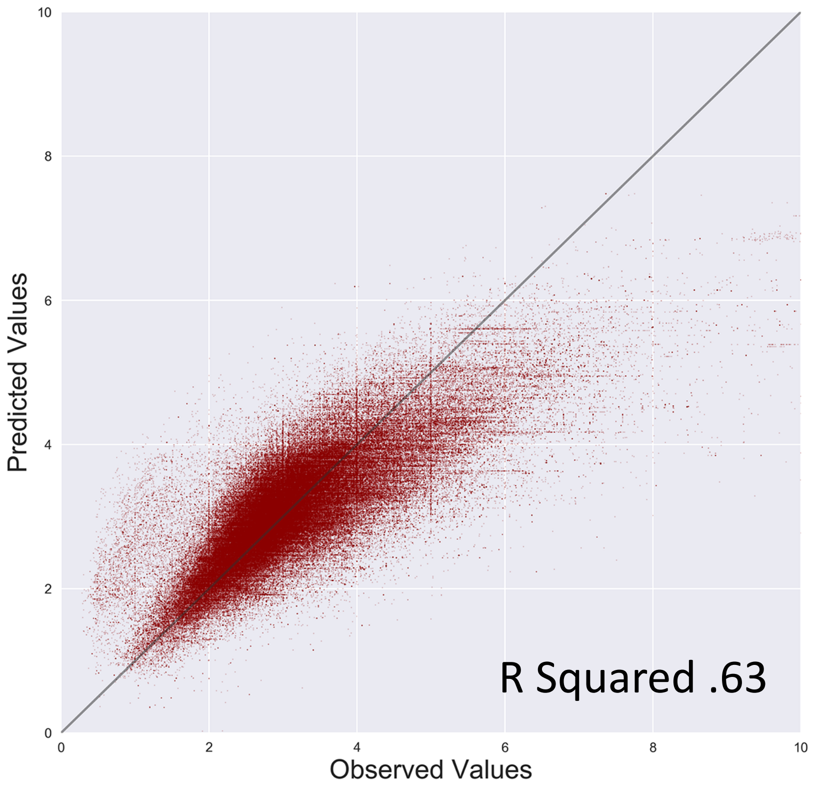
Random Forest



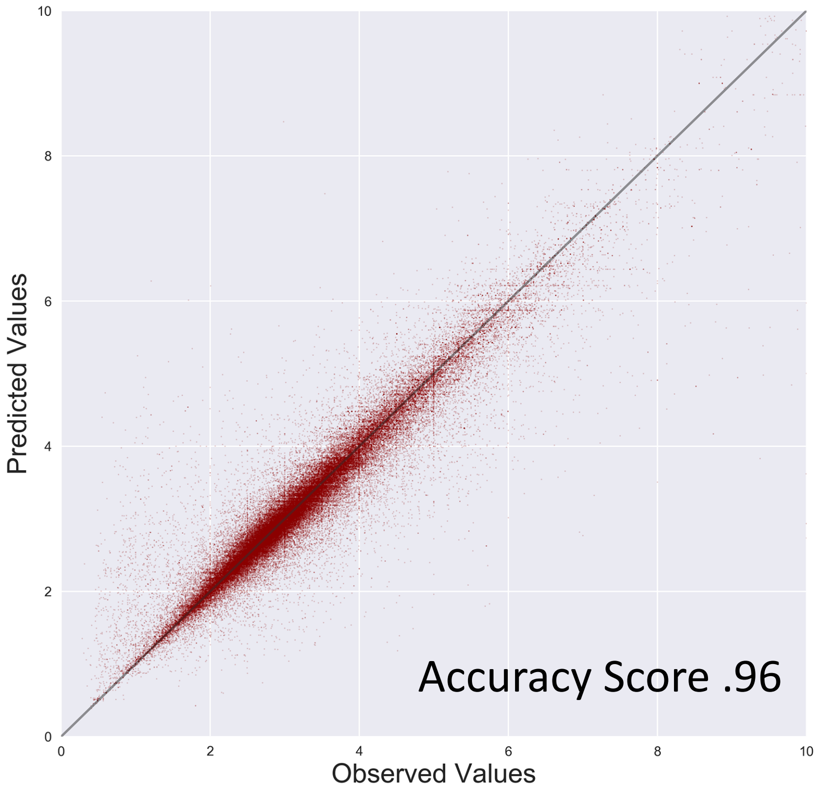
Note: models were estimated using 2/3 of the data, retaining 1/3 for validation

PREDICTED VS OBSERVED

OLS

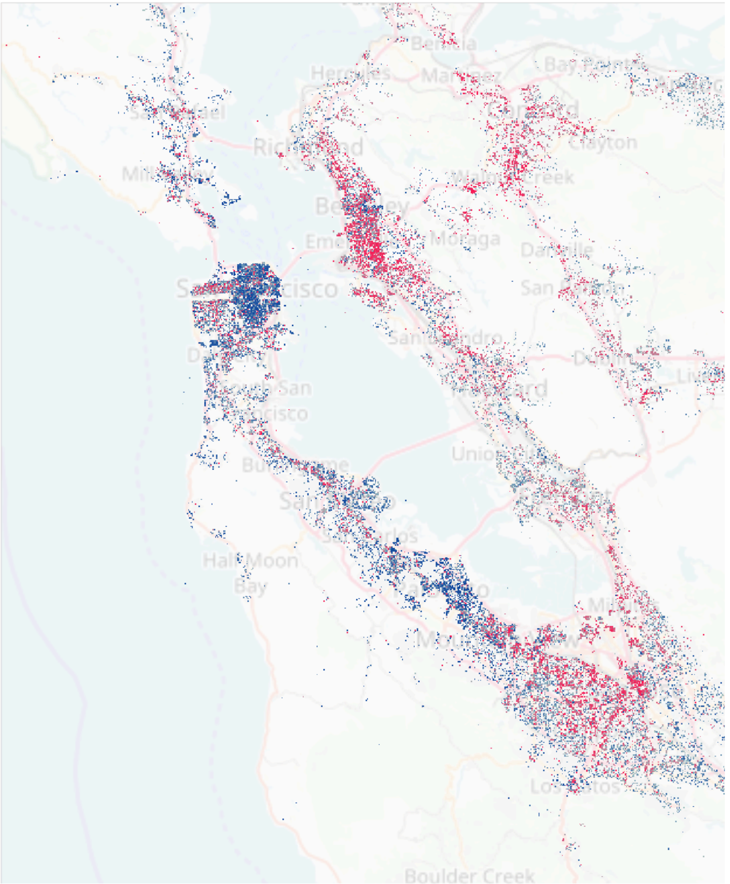


Random Forest

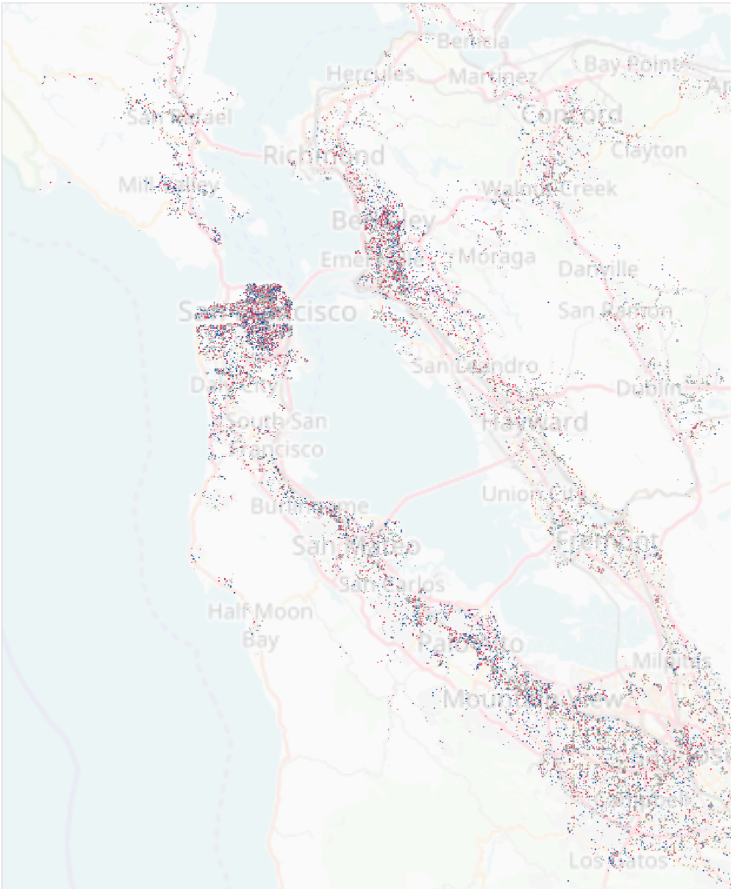


SPATIAL RANDOMNESS OF ERRORS

OLS



Random Forest



Blue areas are under-predicted, red areas are over-predicted

AGENDA

01 ML Hedonic Models to Bootstrap Price Predictions

02 **Differentiable Models for Calibration and Price Equilibration**

03 GPU based Traffic Microsimulation

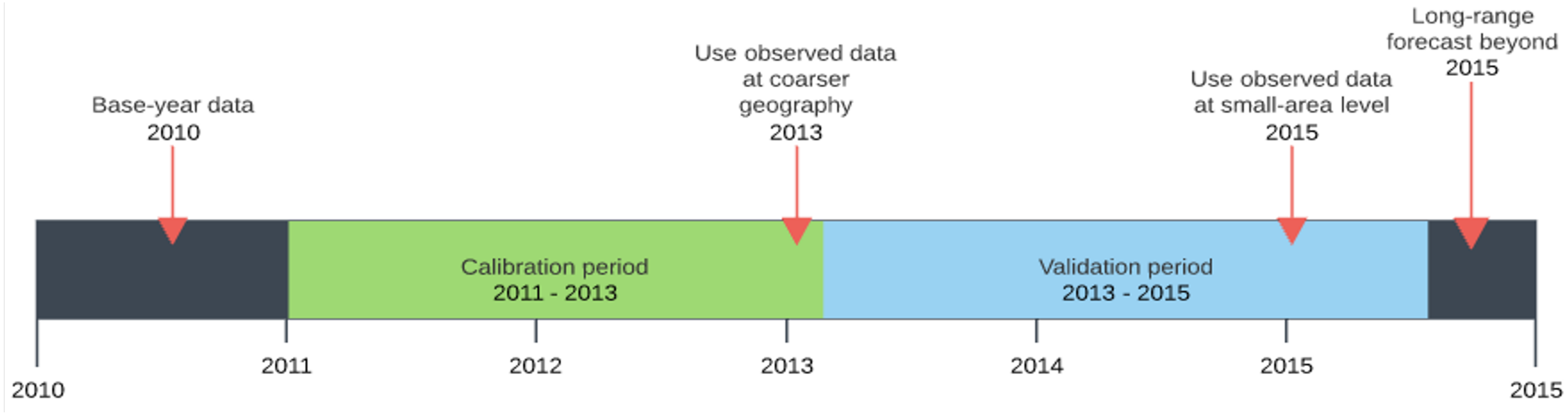
MOTIVATING URBANSIM CALIBRATION

How do we get UrbanSim, a small-area model of urban growth typically estimated off of cross-sectional data, to approximate longitudinal observed data on urban growth?

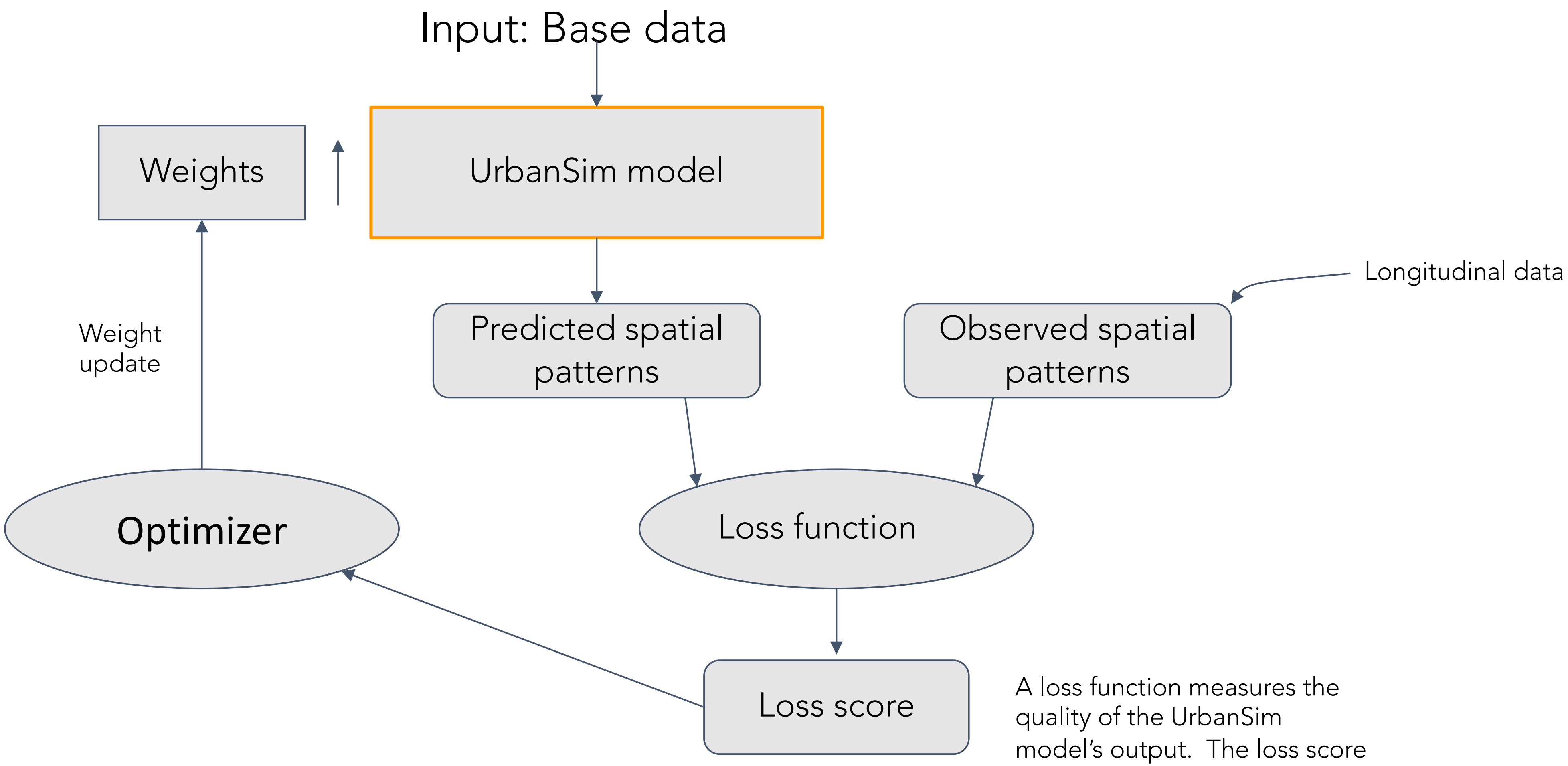
Can we do it without dampening the model’s behavioral sensitivities?
e.g. minimize geographic dummies and constants

TYPICAL GOALS OF MODEL CALIBRATION

- Move relative spatial variation of simulated growth towards observed longitudinal patterns
- Proxy for unobserved costs and variables not accounted for by the models as specified
- Incorporate more information from longitudinal data (model estimation is based on cross-sectional data)



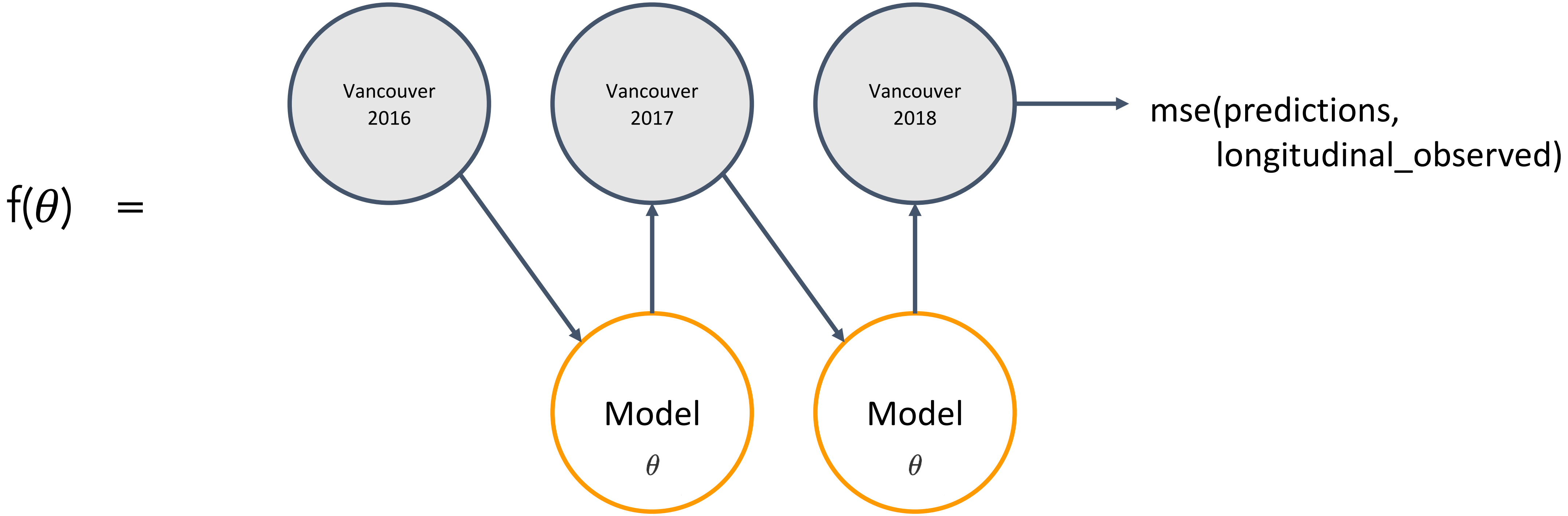
CALIBRATION



A loss function measures the quality of the UrbanSim model's output. The loss score is used as a feedback signal to adjust the weights.

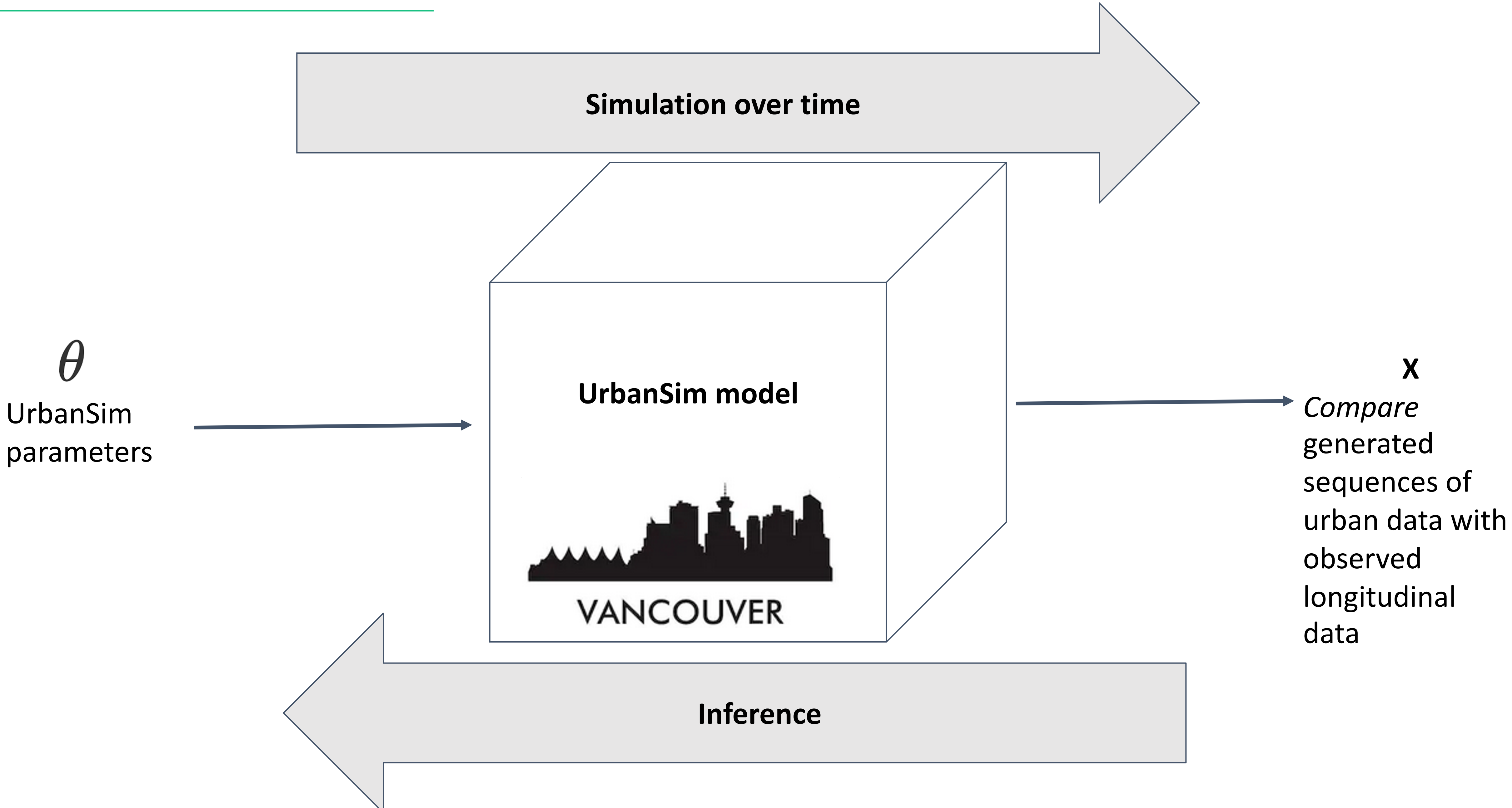
MODEL CALIBRATION

Infer UrbanSim parameters that would minimize error vs. observed longitudinal data.



URBANSIM SIMULATION: “FORWARD MODE”

CALIBRATION

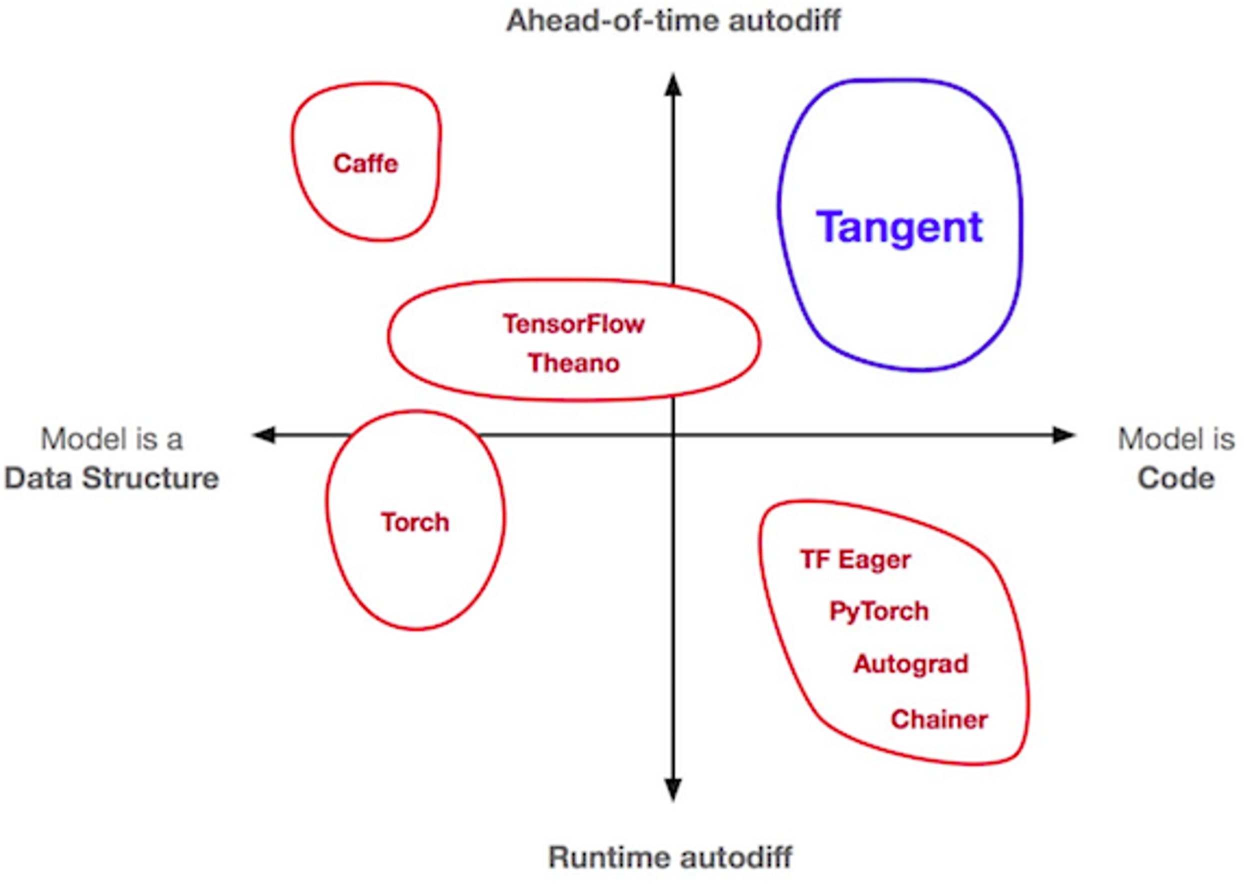


OUR CURRENT APPROACH

- Frame UrbanSim as a differentiable function
- Add objective function and utilize auto-differentiation libraries to get gradients
- Apply gradient descent

How: reimplementing UrbanSim modules in “differentiable programming” frameworks

URBAN MODELS MEET DIFFERENTIABLE PROGRAMMING



Different approaches and styles of modern deep learning libraries.
Not drawn to scale!

TensorFlow

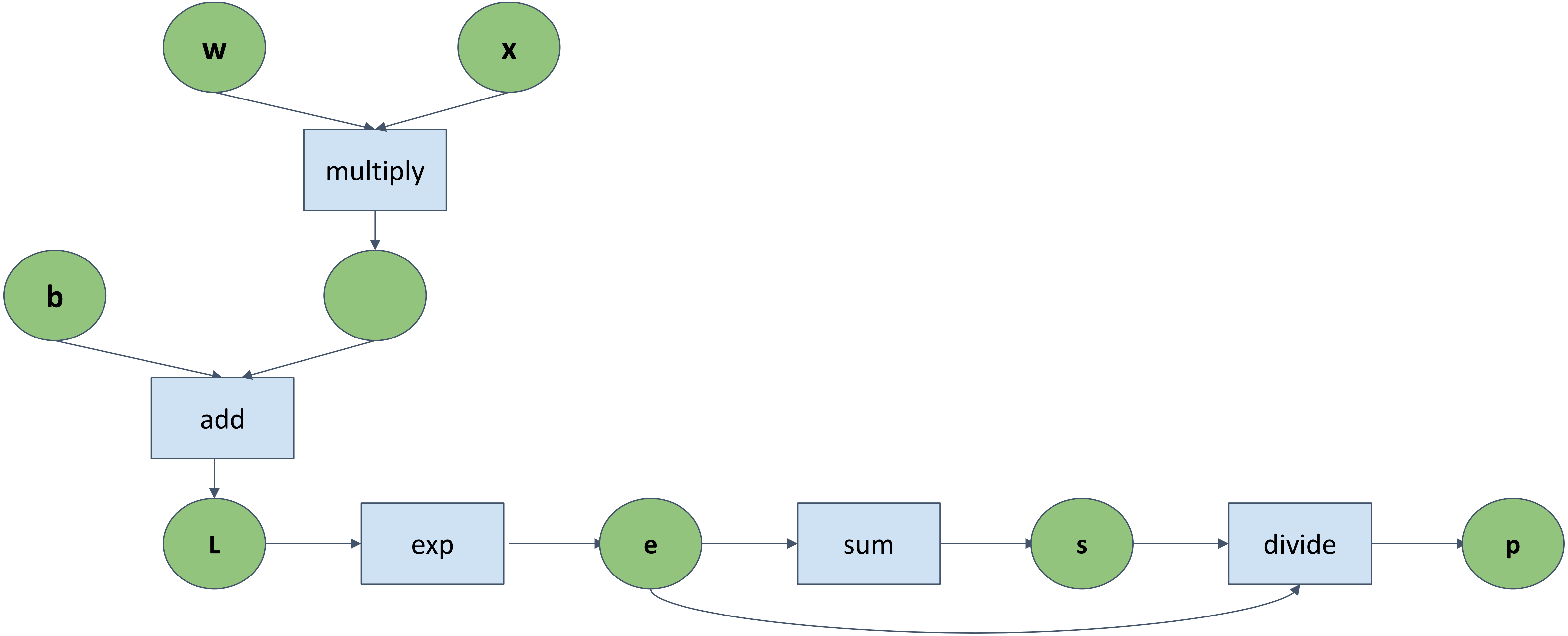
PyTorch

Swift for TensorFlow

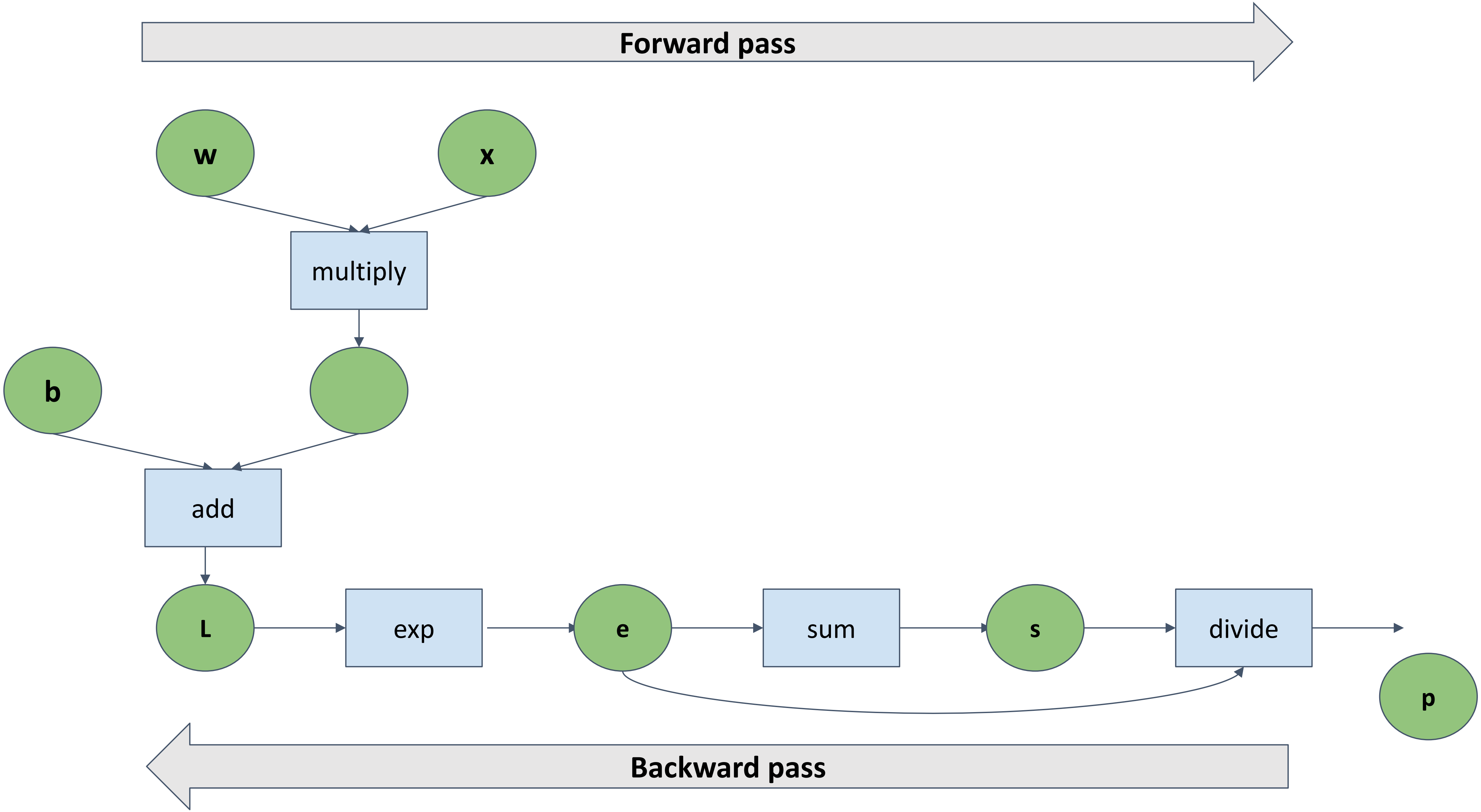
Zygote

EXAMPLE OF A COMPUTATIONAL GRAPH IN URBANSIM

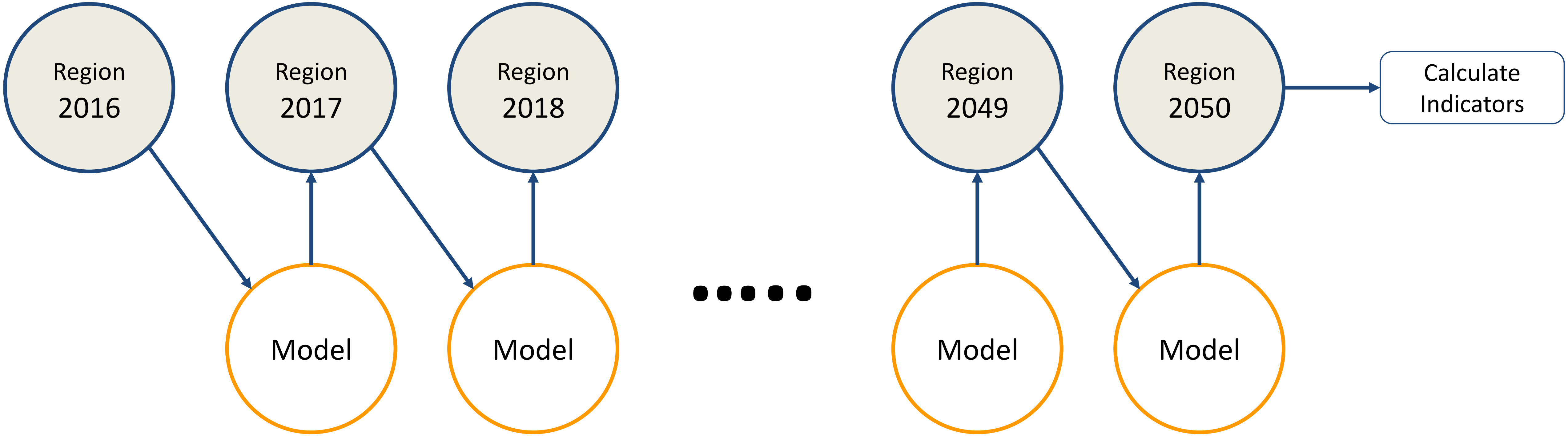
```
logits = np.dot(w, x) + b
exp_utility = np.exp(logits)
sum_expu_across_submodels = np.sum(exp_utility, axis=1, keepdims=True)
probas = exp_utility / sum_expu_across_submodels
```

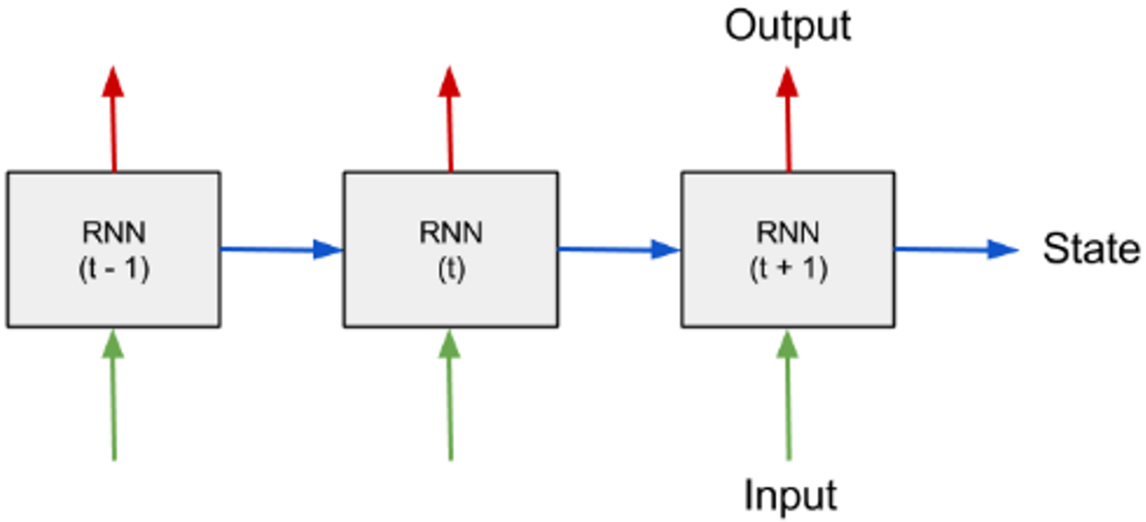


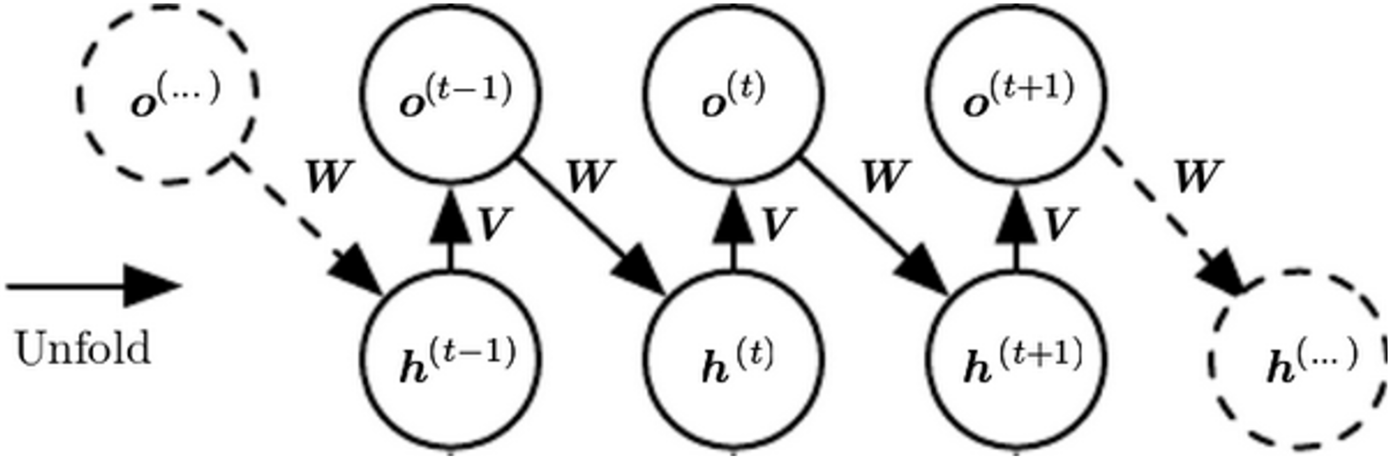
EXAMPLE OF A COMPUTATIONAL GRAPH IN URBANSIM



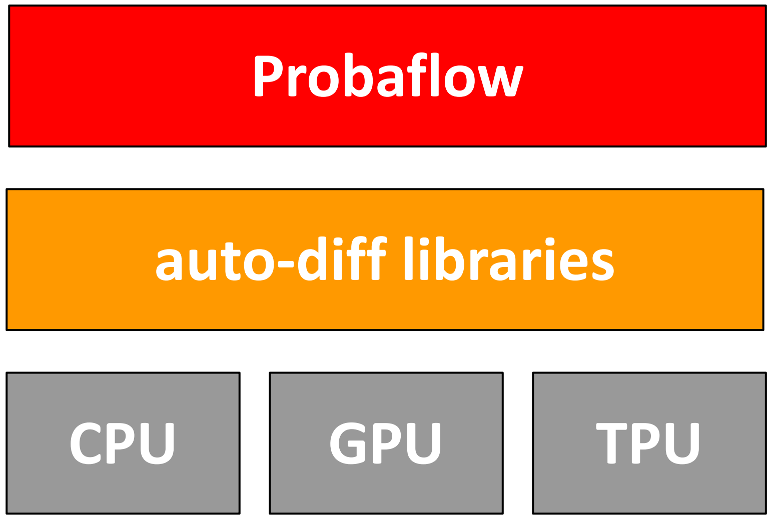
URBAN DYNAMICS AS A RECURRENT SEQUENCE







PROBAFLOW: LIBRARY FOR SPECIFYING/TRAINING DIFFERENTIABLE MODELS



- Some auto-diff libraries take python code and can compile to lower-level representation for hardware acceleration
- Trace gradients through programs, so probaflow-composed models are end-to-end differentiable

PROBAFLOW CAPABILITIES

- Multi-geographic-level optimization. E.g. county + municipality + tract
- Jointly optimize model components over multiple years (better accounting for linkages between models so that overall dynamics are smoother)
- Sign constraints built-in: makes parameter estimation easier
- Regularization default
- Custom loss functions: build desirable properties of the simulation into the loss used to train model
- Multi-region learning + transfer learning

MODEL VALIDATION

STRATEGIES:

- Compare model results to observed data
- Sensitivity tests

LONGITUDINAL VALIDATION (BACKCASTING) EXAMPLE:

Observed tract change:

Change in number of households by tract, 2010 - 2015

Change in number of dwelling units by tract, 2010 - 2015

Change in employment by tract, 2010 - 2015

Change in non-residential square-footage by tract, 2010 - 2015

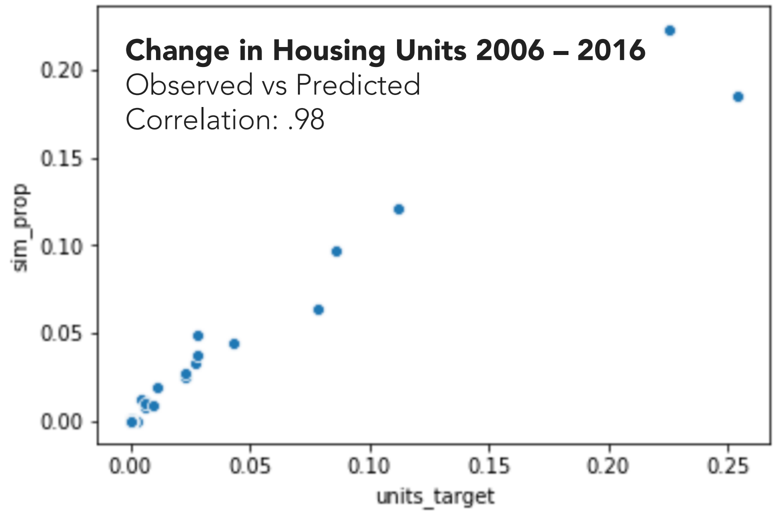
Model base-year: 2010

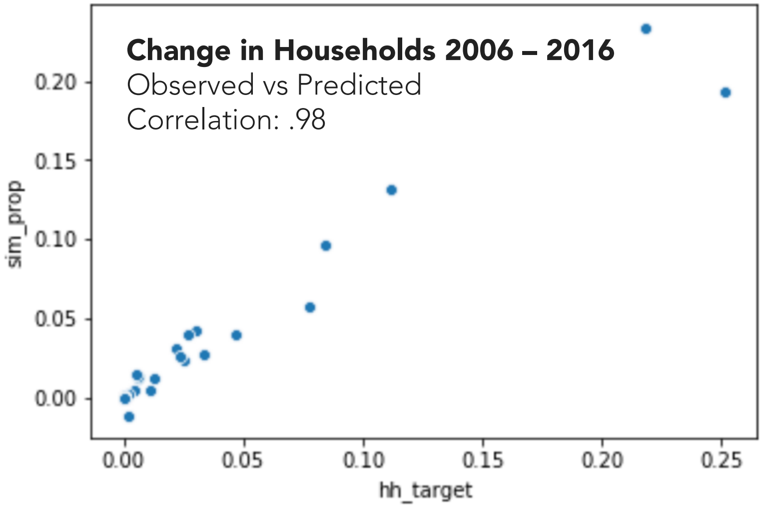
Simulate 2010 - 2015, compare observed tract change vs simulated,

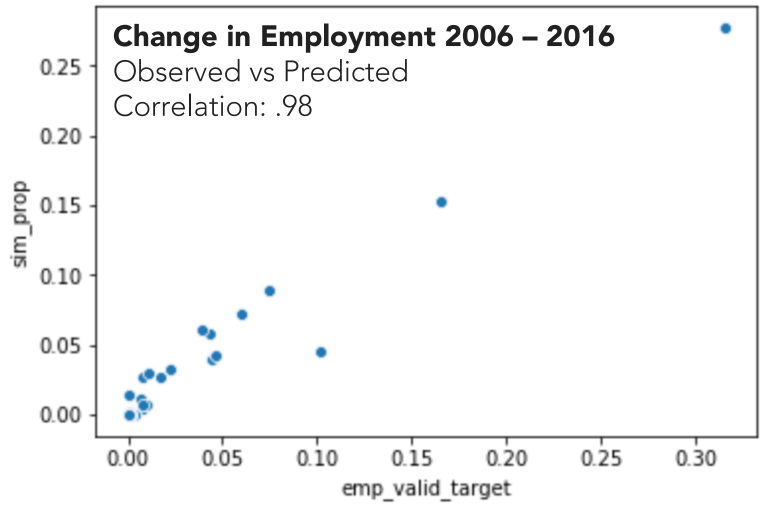
calculate prediction R2 / RMSE

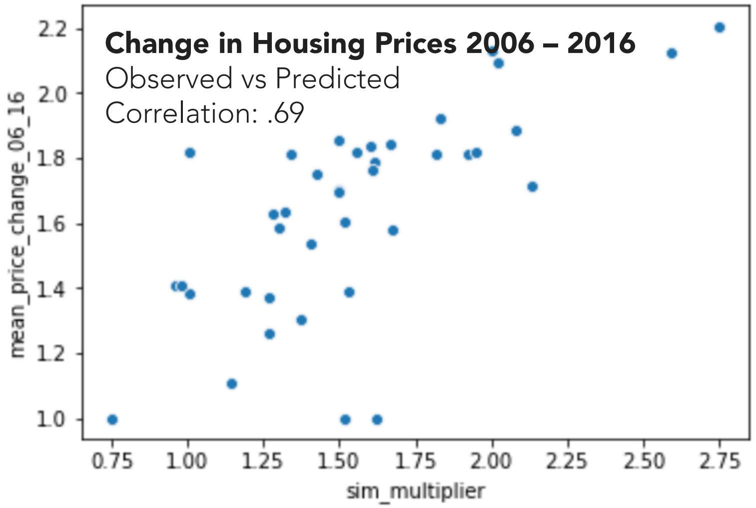
VANCOUVER URBANSIM APPLICATION: VALIDATION

UrbanSim models are validated by running simulations from 2006 – 2016 and comparing simulated to observed data by Census Subdivision









AGENDA

01 ML Hedonic Models to Bootstrap Price Predictions

02 Differentiable Models for Calibration and Price Equilibration

03 **GPU based Traffic Microsimulation**

GPU-BASED TRAFFIC MICROSIMULATION AT METROPOLITAN SCALE

Bay Area network (derived from OSM/OSMnx)

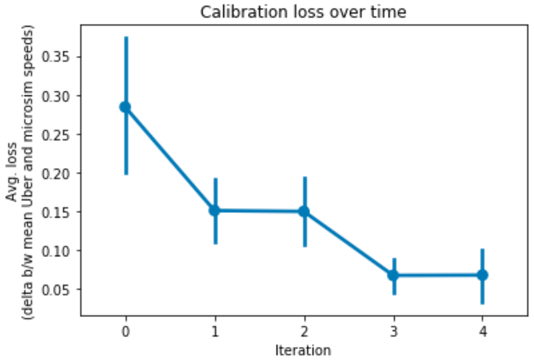
223K nodes

560K edges

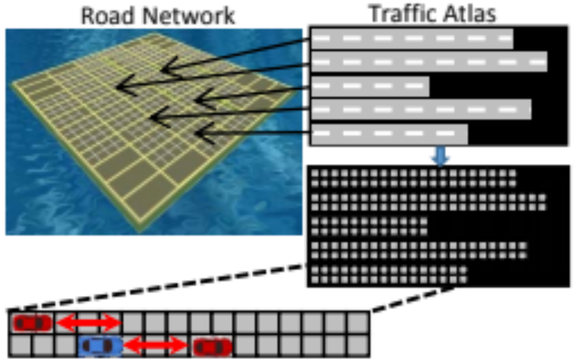
GPU-BASED TRAFFIC MICROSIMULATION AT METROPOLITAN SCALE

Pandana library using Contraction Hierarchies for Fast Computation of Shortest Paths

$$\min_{a,b,T,s_0} \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} [a(1-(\frac{v_k}{v_{0,n}})^{\delta} - (\frac{s_o+Tv+\frac{v}{2\sqrt{ab}}}{s})^2]t}{K} - \overline{v}_{uber}|$$



GPU Traffic Microsimulation



Traffic Atlas.

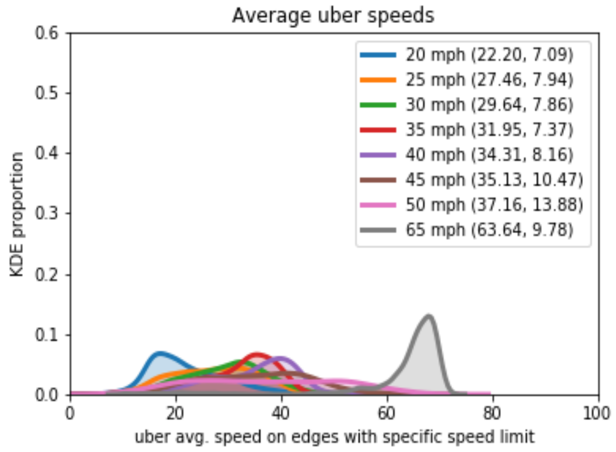
$$\dot{v} = a\left[1 - \left(\frac{v}{v_o}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right]$$

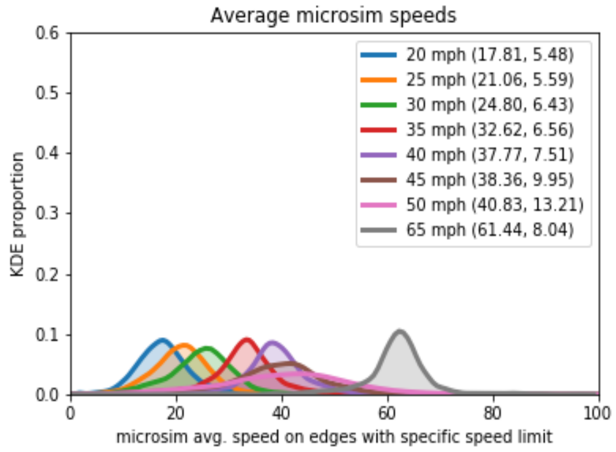
$$m_i = \begin{cases} \exp(-(x_i - x_0)^2) & x_i > x_0 \\ 1 & x_i \leq x_0 \end{cases}$$

Calibration using minibatch gradient descent

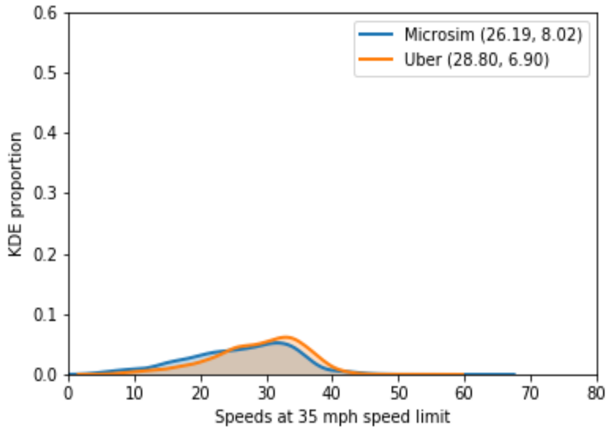
GPU-BASED TRAFFIC MICROSIMULATION: VALIDATION

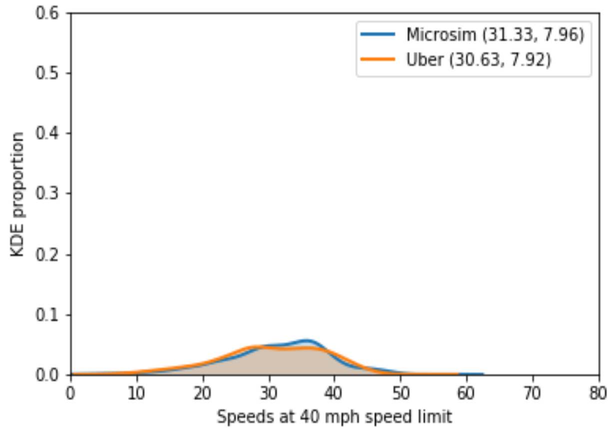
Closely match Uber movement speed data per edge, even with oversimplified intersection traffic controls



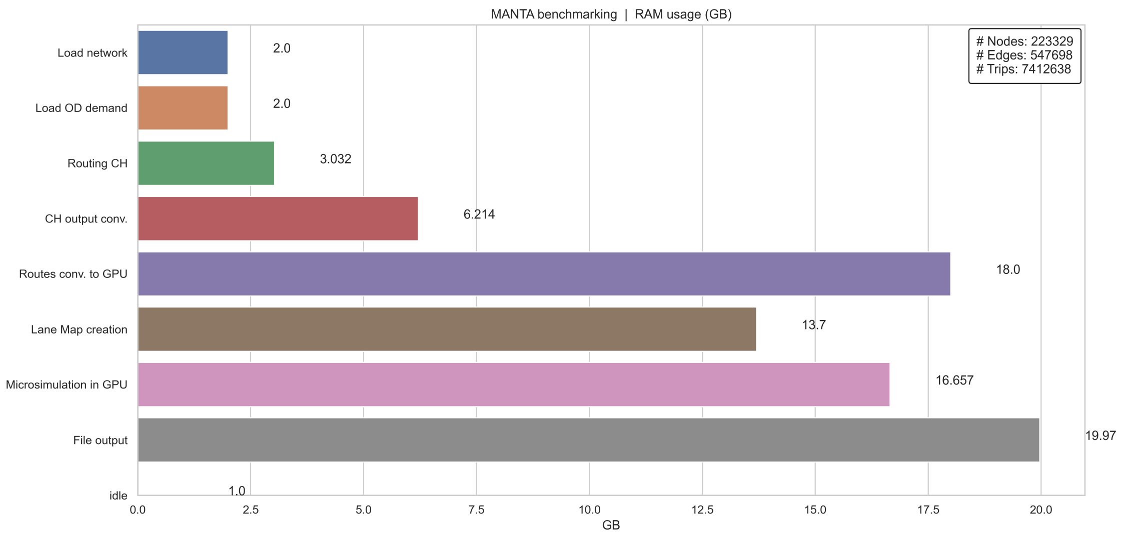


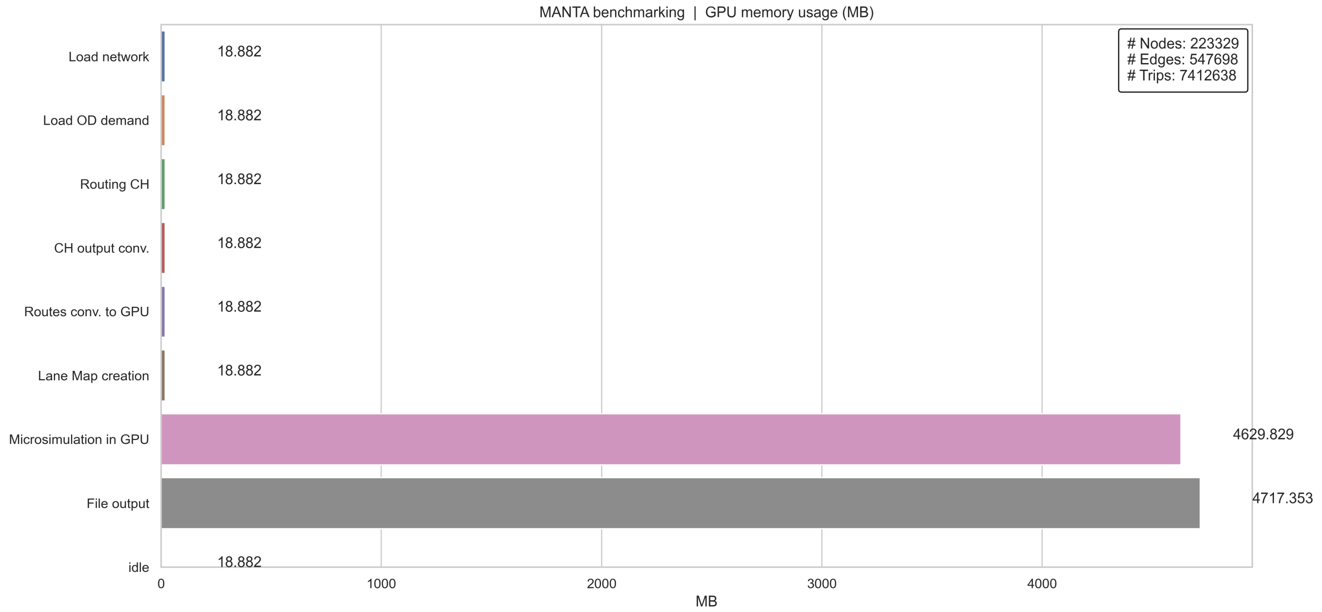
Edge speed limit and Uber standard deviations (2x) used to model Uber distributions more closely



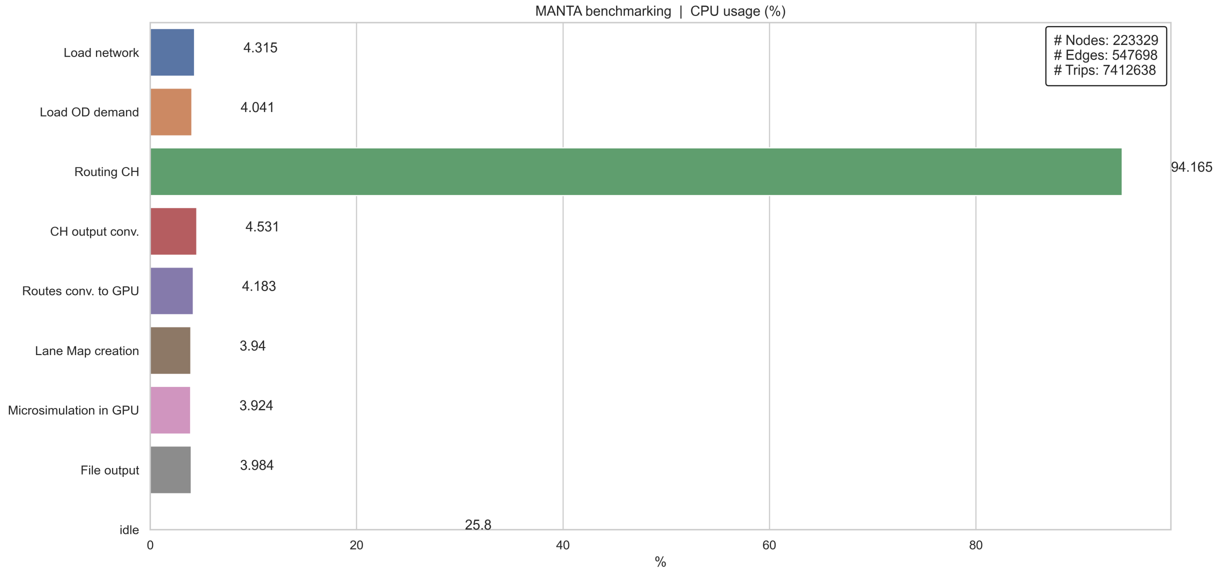


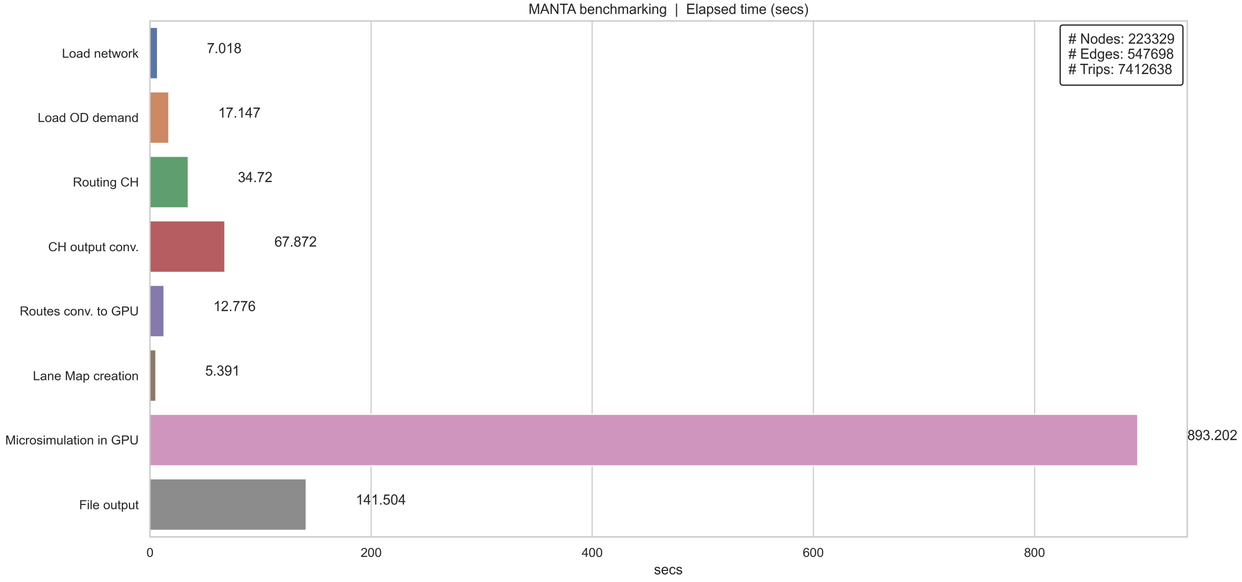
GPU-BASED TRAFFIC MICROSIMULATION: PERFORMANCE





GPU-BASED TRAFFIC MICROSIMULATION: PERFORMANCE





Total: 1179.63 secs (19.66 mins)

CONCLUSIONS

01 Opportunities for leveraging ML to improve urban models

02 Differentiable programming offers strategy to improve longitudinal calibration without loss of sensitivity

03 GPU based Traffic Microsimulation has significant potential to improve realism, performance, scale of microsimulation


Q&A