UVA DATA SCIENCE

Embarrasingly parallel modeling with Lambda

Big Data Systems

Ryan Cox, Griffin McCauley, and Douglas Ziman

May 2nd, 2023

Agenda

Background and approach

Model and pipeline details

Results

Implications

Agenda

Background and approach

Model and pipeline details

Results

Implications

Large machine learning models are becoming a mainstay for business problems, but can be expensive to run

Rise in ML for business

Explosion of big data and better technology has prompted application of machine learning across business contexts such as:

- Product demand prediction
- Customer churn management
- Dynamic pricing schedules

Random Forests are commonly applied

Frequently, these models rely on random forests because they are ubiquitous, easy to explain, and relatively simple

Business requirements exist

These problems are trained on large amounts of data, but training and prediction are needed in near real-time

Cloud costs and runtimes are high

But large prediction trees on big datasets can often lead to high cloud computing costs and runtimes:

- Require many CPUs to speed up tree training
- High memory requirements due to large datasets and tree structures
- Fail to fully parallelize tasks

Solution

Can AWS Lambda functions lower runtime and cloud computing costs for random forest training by leveraging embarrassingly parallel tasks?

Data consisted of 66GB of stock trading data on 100 firms (Nifty 100) gathered at the 1-minute level for 2018-2022

66 GB

Stock 1: NMDC

Dependent variable: Minute-level price data

Independent variables: various moving averages, volatility measures, and other technical trading metrics (approx. 50 columns)

660 MB

Stock 2: RELIANCE

Stock 3: PNB

Stock 100: PGHH

Model idea:

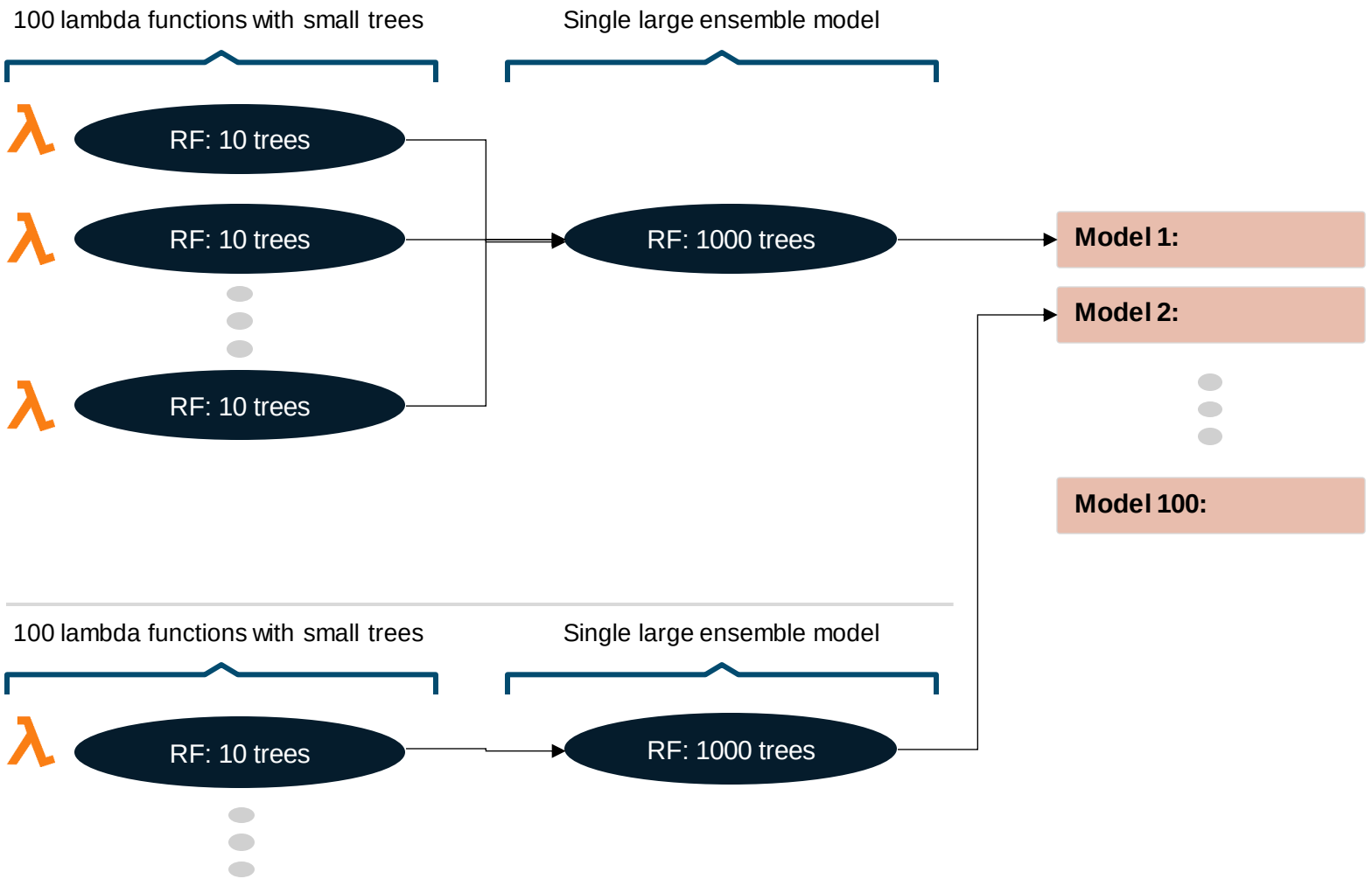
A trading firm may wish to see if past trading data can be used to predict future price movements. Explaining price using time series data is a clear use case.

However, each stock will require a different model, meaning expensive and lengthy training times

Proposal

Parallelize creation of models for **all 100 stocks simultaneously** using AWS Lambda

Random forests are naturally parallel and thus lend themselves well to this approach



100 functions x 100 stocks means potentially thousands of lambda functions running in parallel.

When combined with a parameter grid search on small trees, **we will potentially need tens of thousands** of simultaneous functions

Agenda

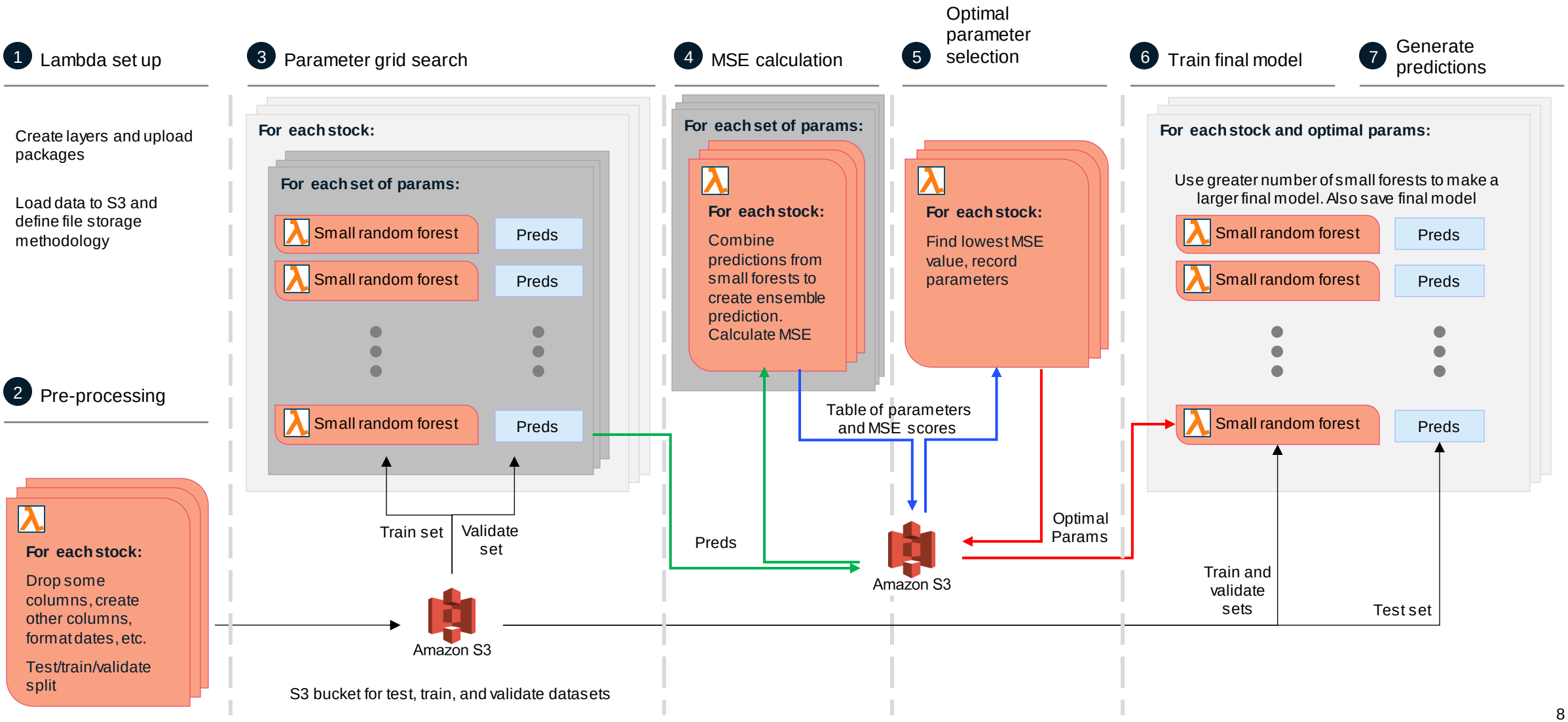
Background and approach

Model and pipeline details

Results

Implications

A fully parallel approach requires a sequence of lambda functions passing intermediate steps to S3 storage



1 Lambda set up

Setup and configuration

1. Create IAM role with relevant permissions for S3, Lambda, and CloudWatch (for metrics)
2. Give Lambda access to necessary python packages including sklearn and pandas
3. Design file storage system and load data into S3 buckets
4. Configure Lambda functions with python 3.9, x86_64, max memory size (3000MB), max time limit (15 min), zero retries on failure and 1 hour max in asynchronous queue
5. Increase concurrency limit from default 10 to 1000

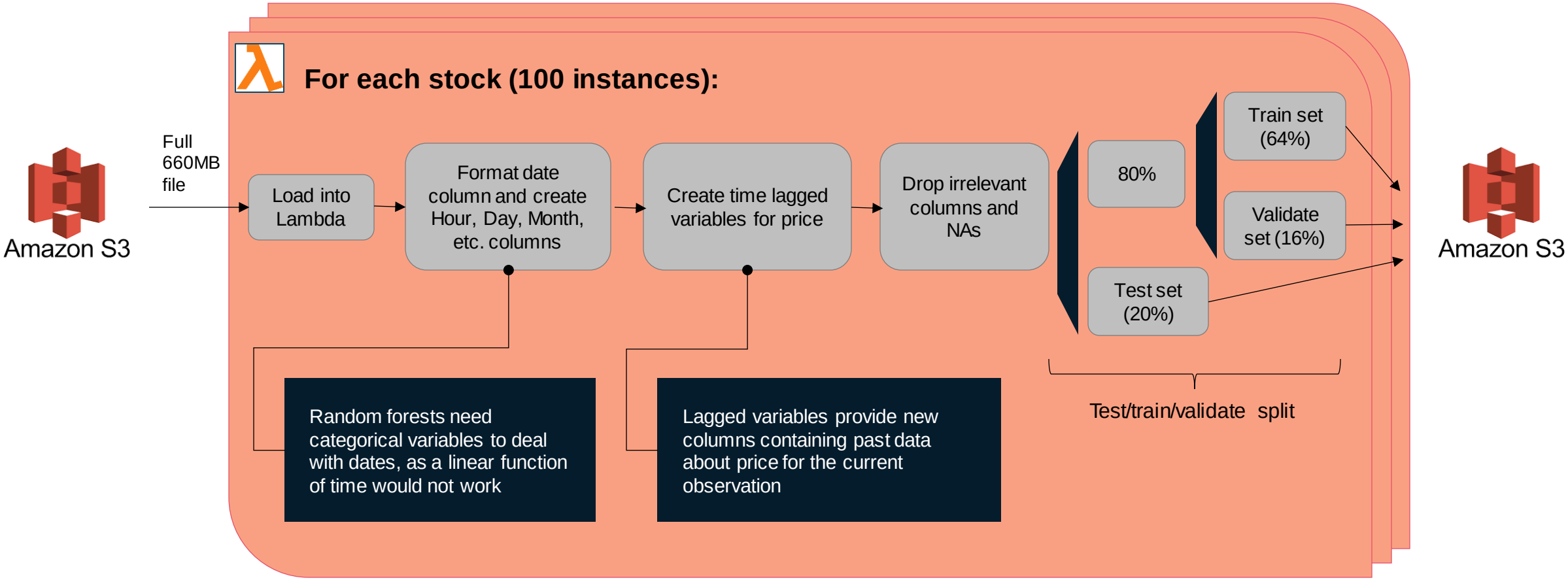
Limitations of AWS Lambda

Lambda only has the ability to load 250MB of code layers. This limits the number of available python packages for any one lambda function.

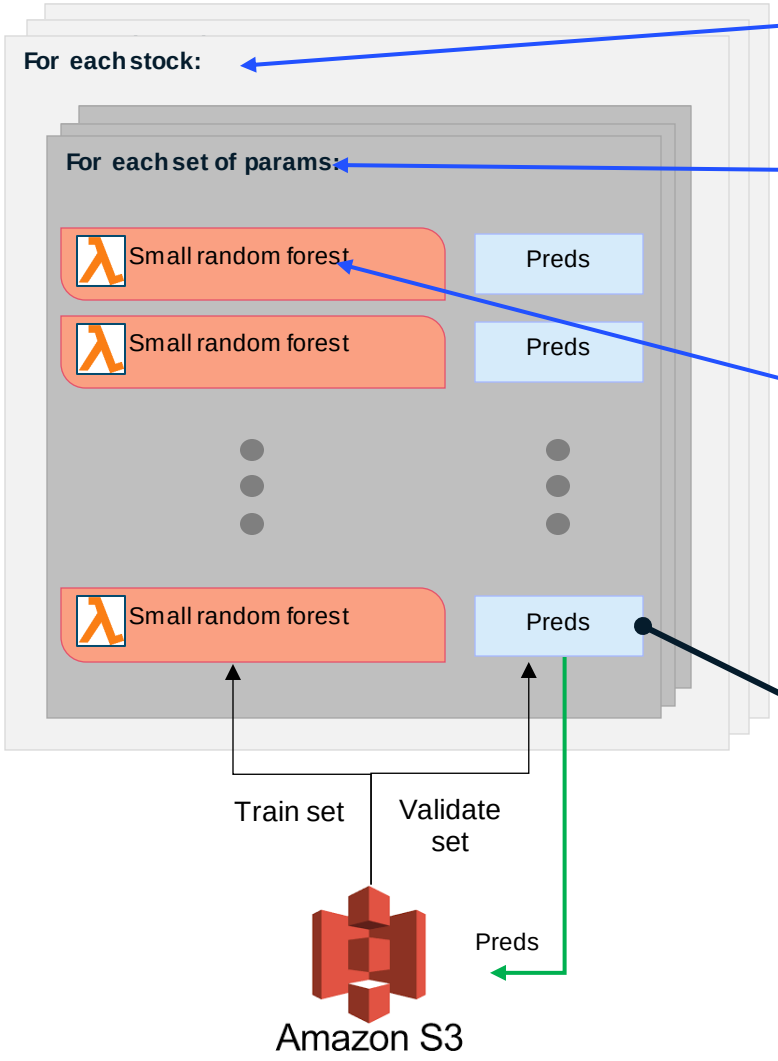
Memory limits became a challenge because even small random forests on this quantity of data resulted in large model files. We had to use creative solutions to avoid running into memory issues.

AWS limits concurrency by default. Presumably, research and corporate accounts could have access to more concurrency.

2 Preprocessing



3 Parameter search and prediction generation



Each stock (x100) can be started in parallel

100x

Parameter grid consisted of 3 values of n_feat (number of features) and min_samp (minimum number of samples in each leaf node) for a total of 9 models for each stock. This could be expanded arbitrarily.

9x

Each random forest consisted of 100 trees. For training we used three lambda functions for each param set for a total of 300 trees

3x

2700

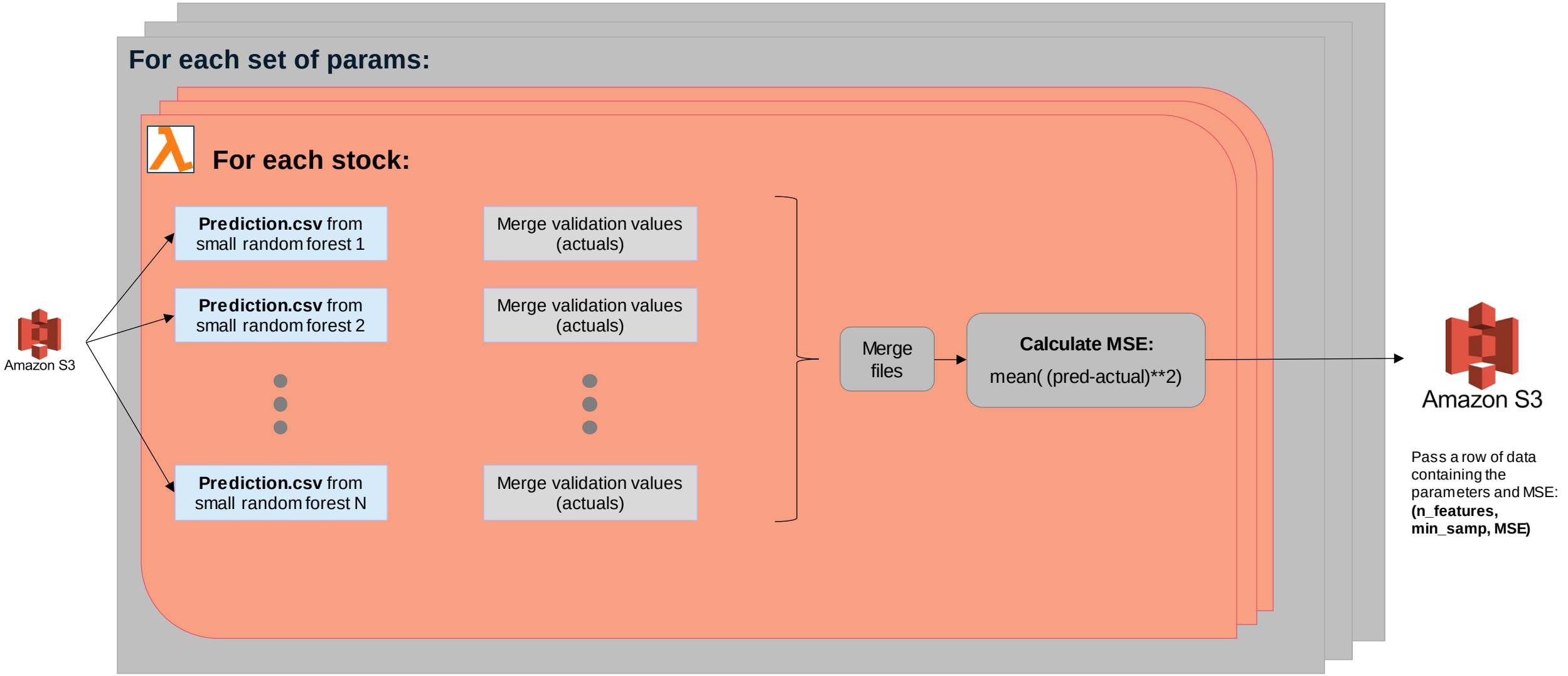
Theoretical number of parallel lambda functions (with no AWS concurrency throttle)

Prediction approach

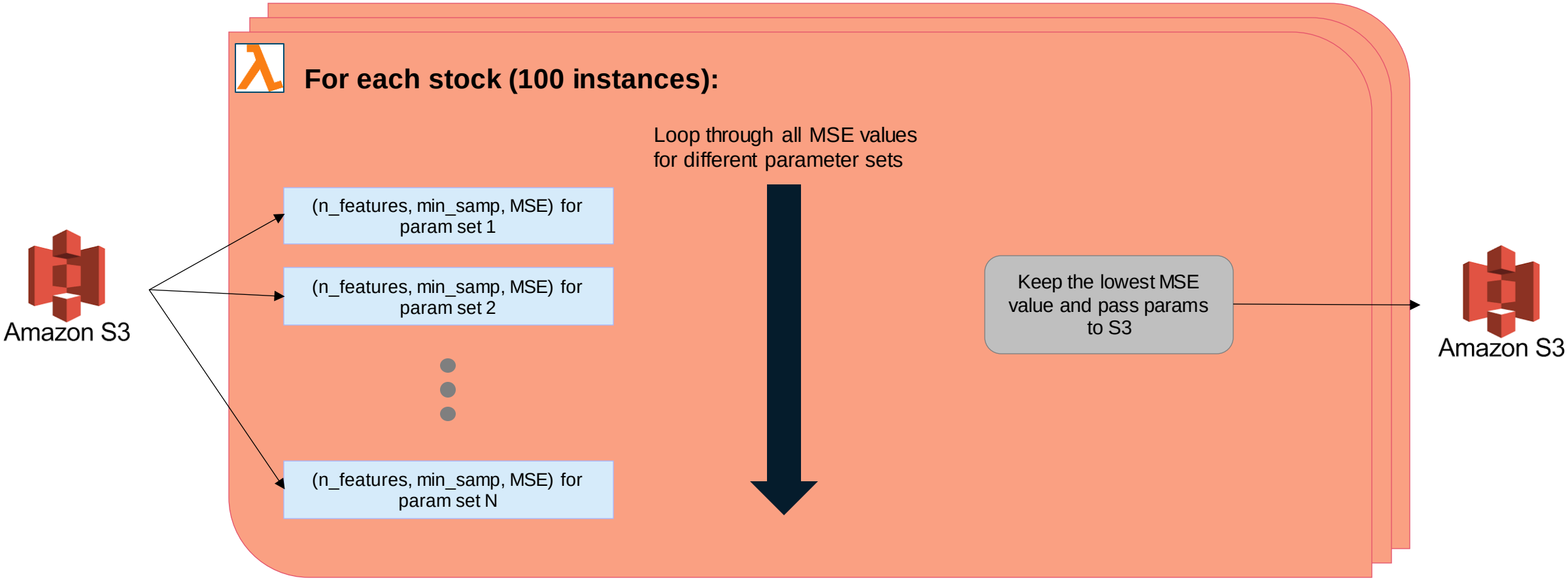
Our original plan was to save each **model** and sum them up (trivial with forests). However, models became too large to be reloaded into lambda

Instead, we save the predictions on the validation set back to S3 (single column of values for each row in the validation set) which results in 2700 small files from which an MSE can be generated

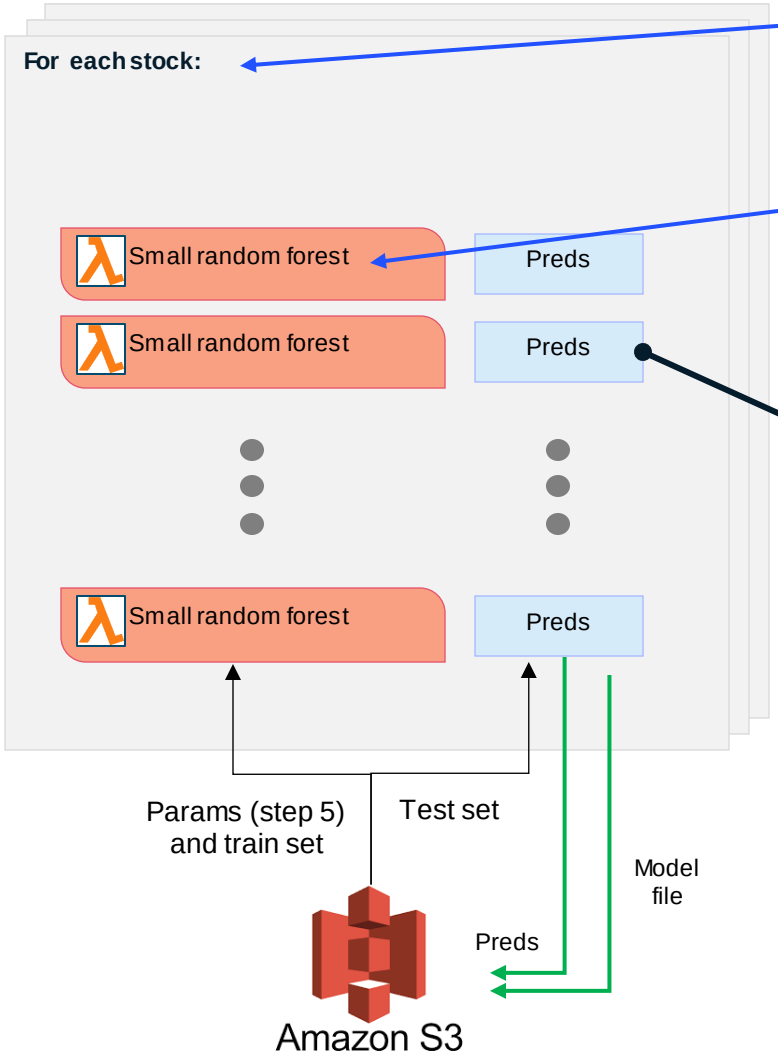
4 Calculation of MSE values for parameter optimization



5 Find minimum MSE



6 Train final model with best parameters



Each stock (x100) can be started in parallel

100x

Each random forest consisted of 25 trees. For final models we used 30 lambda functions for each param set for a total of 750 trees

30x

Prediction approach

In this case we generated the same distributed set of predictions from each small forest, but additionally we saved the model files so they can be reloaded later.

In production, these files could be aggregated easily in a EC2 instance for simpler usage

3000

Theoretical number of parallel lambda functions (with no AWS concurrency throttle)

7 Test model

To perform testing, we implemented the same infrastructure as can be seen in step 4. This testing can be done in parallel with one lambda per stock.

I/O is high for this function because each one must read 30 separate prediction files, or 3000 simultaneous file reads for a fully parallel run. This was not a major inhibitor of performance.

The theoretical architecture was the focus of the project, so we do not report MSE values

Agenda

Background and approach

Model and pipeline details

Results

Implications

Results show that Lambda approach can train all models in a fraction of the time required by a dedicated machine

Piece of Functionality	Lambda Invocation (Time in sec)	Lambda Computation (Time in sec)	Lambda I/O (Time in sec)	Lambda Fully Parallel Total (Time in sec)	Lambda Throttled Total (Time in sec)	4 core, 16 GB EC2 Total (Time in sec)
Preprocessing	3.3	35.5	24.4	63.2		2,298
Train trees across parameter grid	84.4	529.7	4.5	618.7		187,892
Calculate grid MSEs	3.5	0.3	9.9	13.8		
Select best MSE	3.5	0.1	1.5	5.0		
Train full model	92.4	151.1	6.9	250.5		68,969
Predict for full model	3.4	0.5	13.3	17.2		16.44
Total runtime				**968.3**	**2,337.6**	**259,176**

Lambda fully parallel assumes maximum runtime for lambda functions for parallelization purposes
Lambda throttled was run during experiment but was capped at 1000 concurrent requests
4 core, 16 GB assumes sequential run extrapolating results from single representative stock

110X

Lambda could be over 100 times faster than a normal sized laptop or EC2

Agenda

Background and approach

Model and pipeline details

Results

Implications

Lambda's immense runtime improvement over EC2 does come with drastically increased cloud costs

Architecture	Instance Type	Runtime (minutes)	Estimated Cost
Lambda with project configuration (3008 MB)	NA	39	$53.89
Lambda with maximum capabilities (10 GB, no concurrency limits)	NA	16	$158.11
EC2	t4g.xlarge (4 core, 16GB)	2,580	$5.79
EC2	m6g.4xlarge (16 core, 64GB)	646	$6.63
EC2	c6a.48xlarge (192 core, 384GB)	54	$6.59

Maximum Lambda and EC2s were priced via AWS cost estimator and assumed single run of model scaled to corresponding number of cores

EC2 parallelization would require use of async.io and significant infrastructure setup