KAGGLE MEETUP
XGBOOST VS LIGHTGBM

Algolia kaggle

SPEAKERS

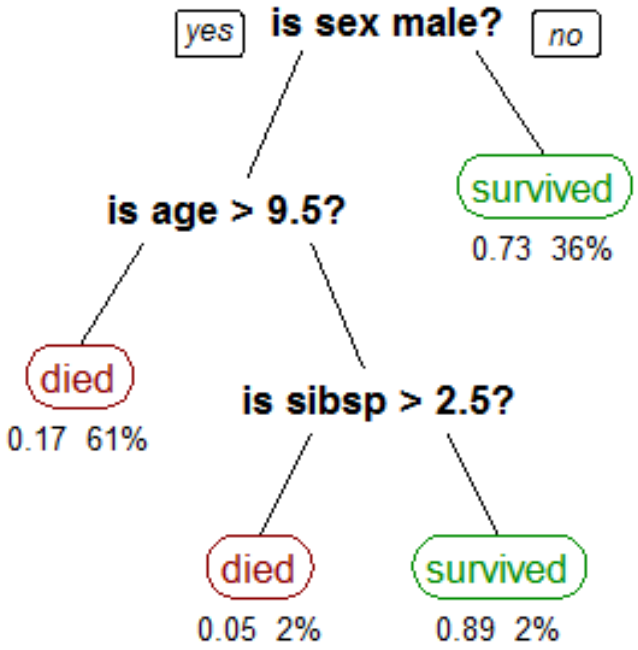
- Damien Soukhavong (*Laurae*)
  - *Data Scientist* at random time
  - *UX Designer* at random time

- Mamy Ratsimbazafy (*mratsim*)
  - *Data Scientist* at night

KEY TAKEAWAYS

- Decision Trees vs Random Forest vs Gradient Boosted Trees
- Be fast
- Be ~~furious~~ accurate

DECISION TREES HISTORY



- Decision Trees
- Random Forest
- Gradient Boosted Trees
- xgboost / LightGBM (Quasi-Newton)

Algolia kaggle

XGBOOST IS WINNING

Algolia kaggle

SPEED OF ALGORITHMS

- caret/scikit-learn < H2o < xgboost exact < LightGBM / xgboost fast histogram

WORST
"days"

GREAT
"hours"

BEST
"minutes"

Algolia

kaggle

PLAN

- Comparing xgboost / LightGBM
  - All you want to know about hyperparameters
- Compilation Flags / Compiler
- Multithreading: tips
- Deep Forest: a failing CNN alternative?

Algolia kaggle

FAST HISTOGRAM XGBOOST?

Histogram Optimized Tree Grower #1940

Merged tqchen merged 17 commits into `dmlc:master` from `unknown repository` on Jan 13

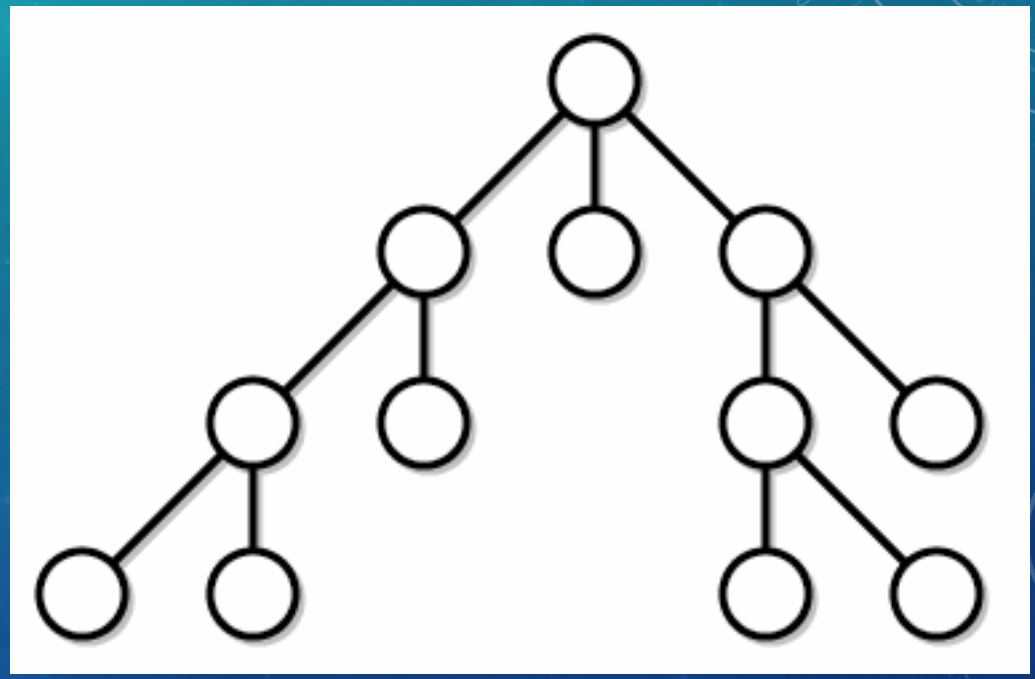
Improve multi-threaded performance #2104

Merged tqchen merged 14 commits into `dmlc:master` from `hcho3:master` on Mar 25

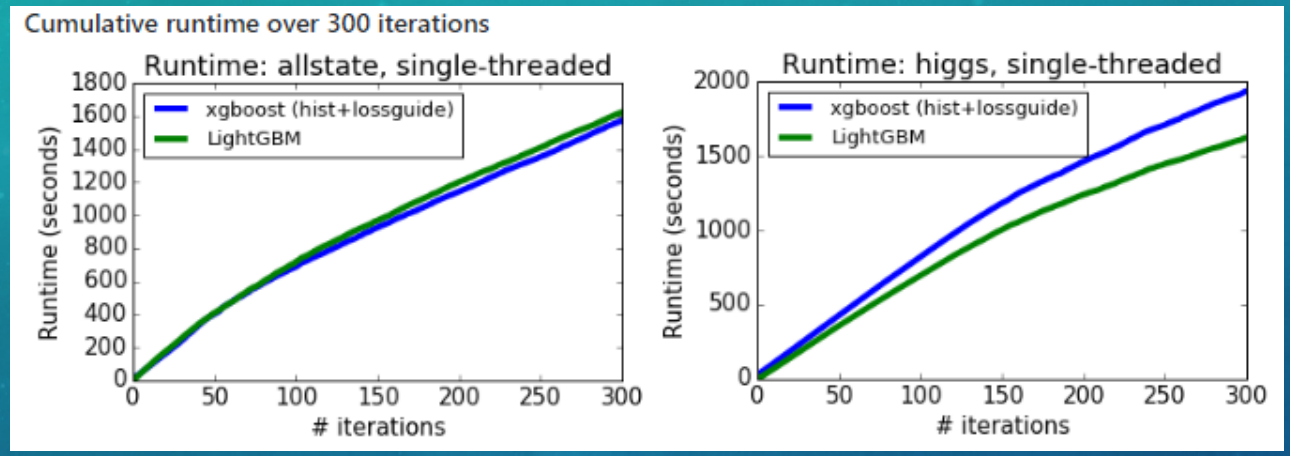
Algolia kaggle

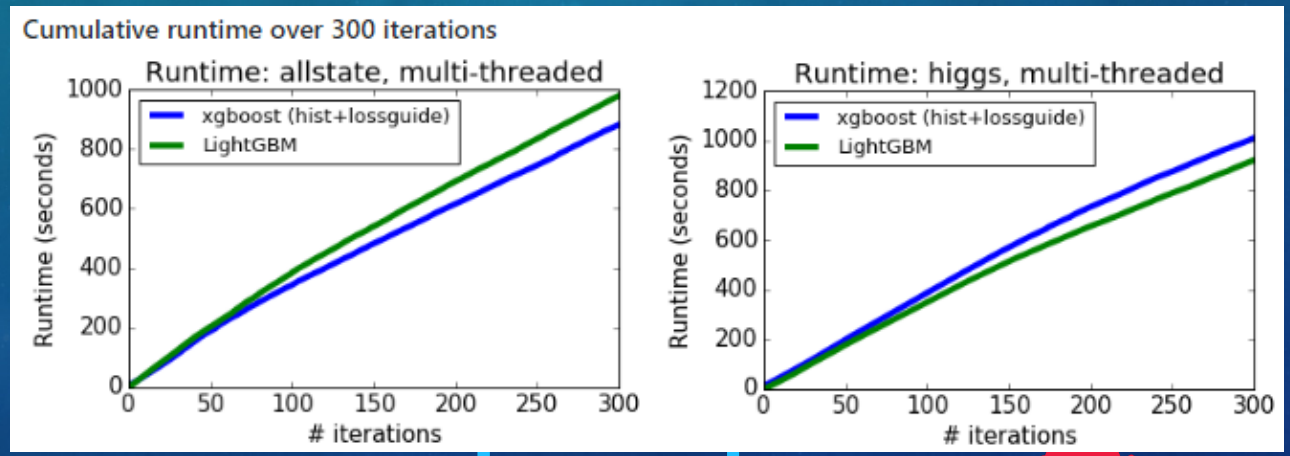
FAST HISTOGRAM XGBOOST?

- Two methods:
  - Depth-wise building
  - Loss guide building



FAST HISTOGRAM XGBOOST?





LIGHTGBM?

- Loss guide method for building trees

Data	xgboost	xgboost_hist	LightGBM
Higgs	3794.34 s	551.898 s	238.505513 s
Yahoo LTR	674.322 s	265.302 s	150.18644 s
MS LTR	1251.27 s	385.201 s	215.320316 s
Expo	1607.35 s	588.253 s	138.504179 s
Allstate	2867.22 s	1355.71 s	348.084475 s

Higgs's AUC:

Metric	xgboost	xgboost_hist	LightGBM
AUC	0.839593	0.845605	0.845154

auc at Expo:

Metric	xgboost	xgboost_hist	LightGBM
auc	0.756713	0.777777	0.777543

auc at Allstate:

Metric	xgboost	xgboost_hist	LightGBM
auc	0.607201	0.609042	0.609167

Algolia kaggle

COMPARING WHAT?

- Speed & Performance
  - *"Exact"* xgboost
  - xgboost fast histogram method
    - → Depthwise & Lossguide
  - LightGBM v1 and v2
- Compilation flags for speed

XGBoost

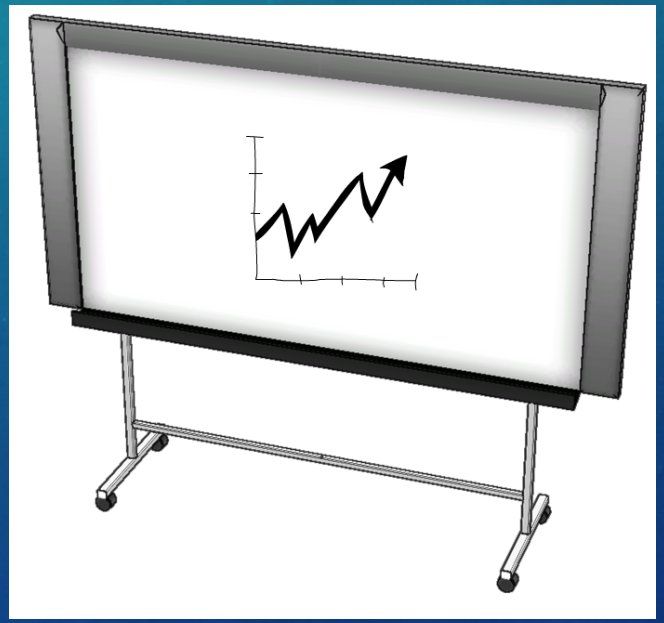
LightGBM

Algolia kaggle

SPEED & PERFORMANCE?

Speed

Performance



Algolia kaggle

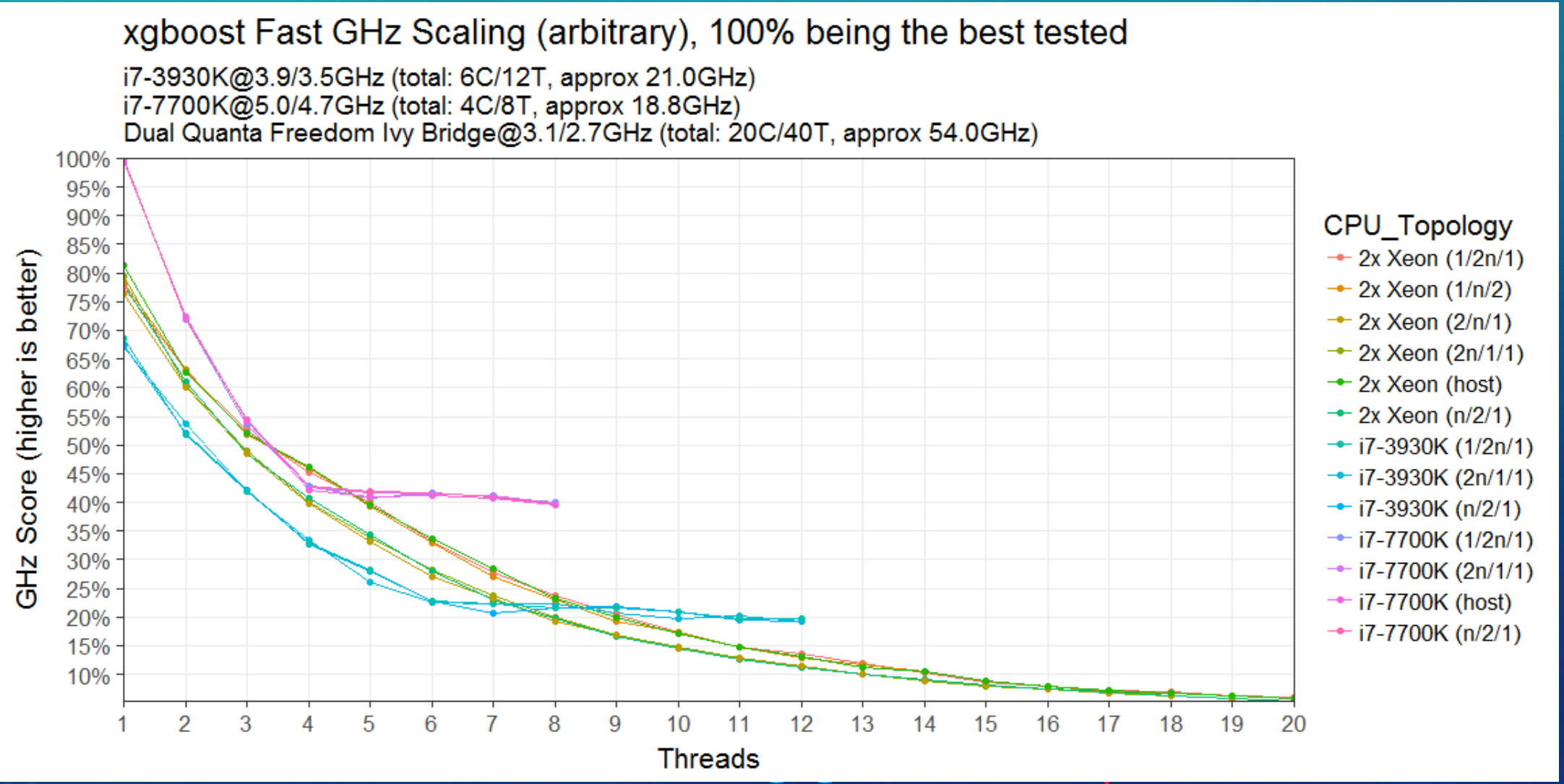
ENVIRONMENT USED

- OS: Windows Server 2012 R2
- CPU / Virtualization:
  - VMware: Intel i7-3930K for 1-6 + 12 threads
  - KVM: Dual Quanta Freedom Ivy Bridge for 20-40 Threads

- VMware : expect 25%+ performance on baremetal
- KVM: expect 10%+ performance on baremetal

ENVIRONMENT USED

i7-3930K vs i7-7700K (almost identical)



PROGRAMMING SETUP

- Programming language:
  - R 3.3.2: main horse
  - bash: spawn Rscript code with arguments
  - CLI: get exact RAM usage during training

- Expect ~0% gain using Python
- Expect ~0% gain using CLI

Algolia kaggle

DATASETS

- Noisy ⟶ Bosch:
  - From Kaggle / Bosch / Ieee Bigdata 2016
  - 1,000,000 for training
  - 970 features

DATASETS

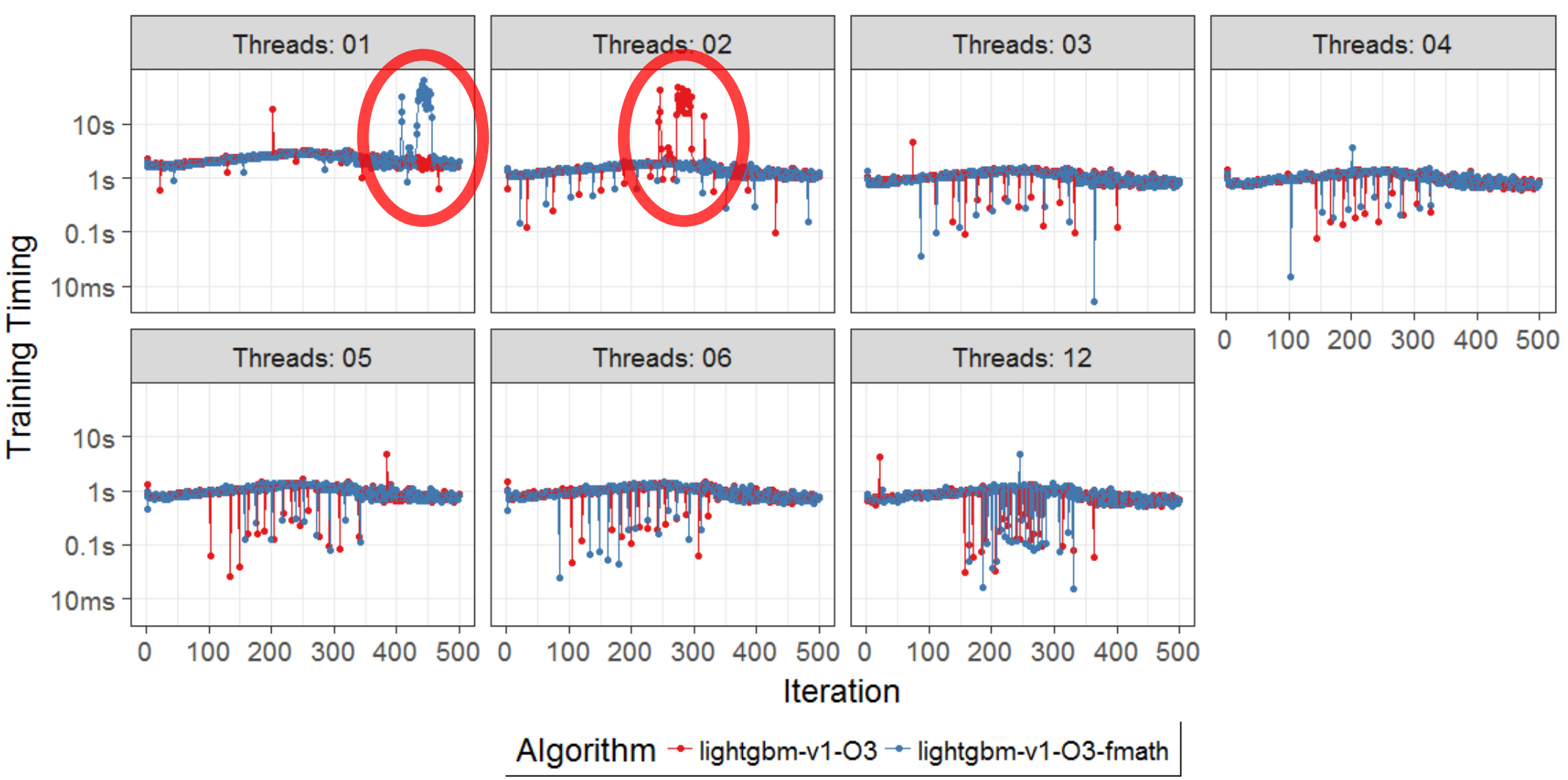
- Synthetic → Higgs:
  - From University of California Irvine
  - 10,000,000 for training
  - 29 features

SPECIAL FINDINGS

- LightGBM v1 or v2: **not deterministic** when Stochastic
- xgboost: difficulties converging on Higgs
- Still comparing unfairly ⟶ stop at the first iteration they stop learning (deleted: Higgs)

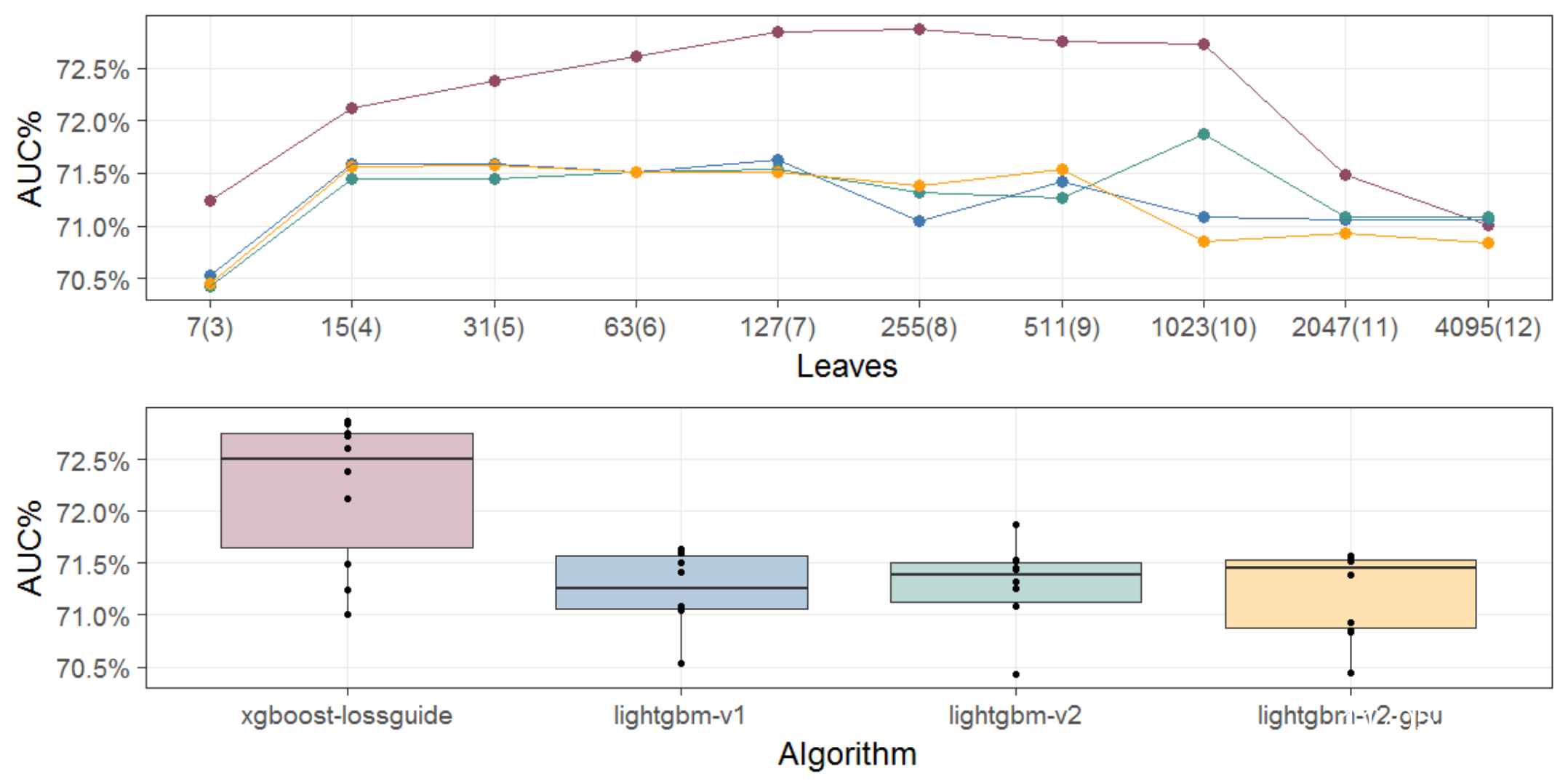
SPECIAL FINDINGS

Weird peaks (Bosch, Depth 7)



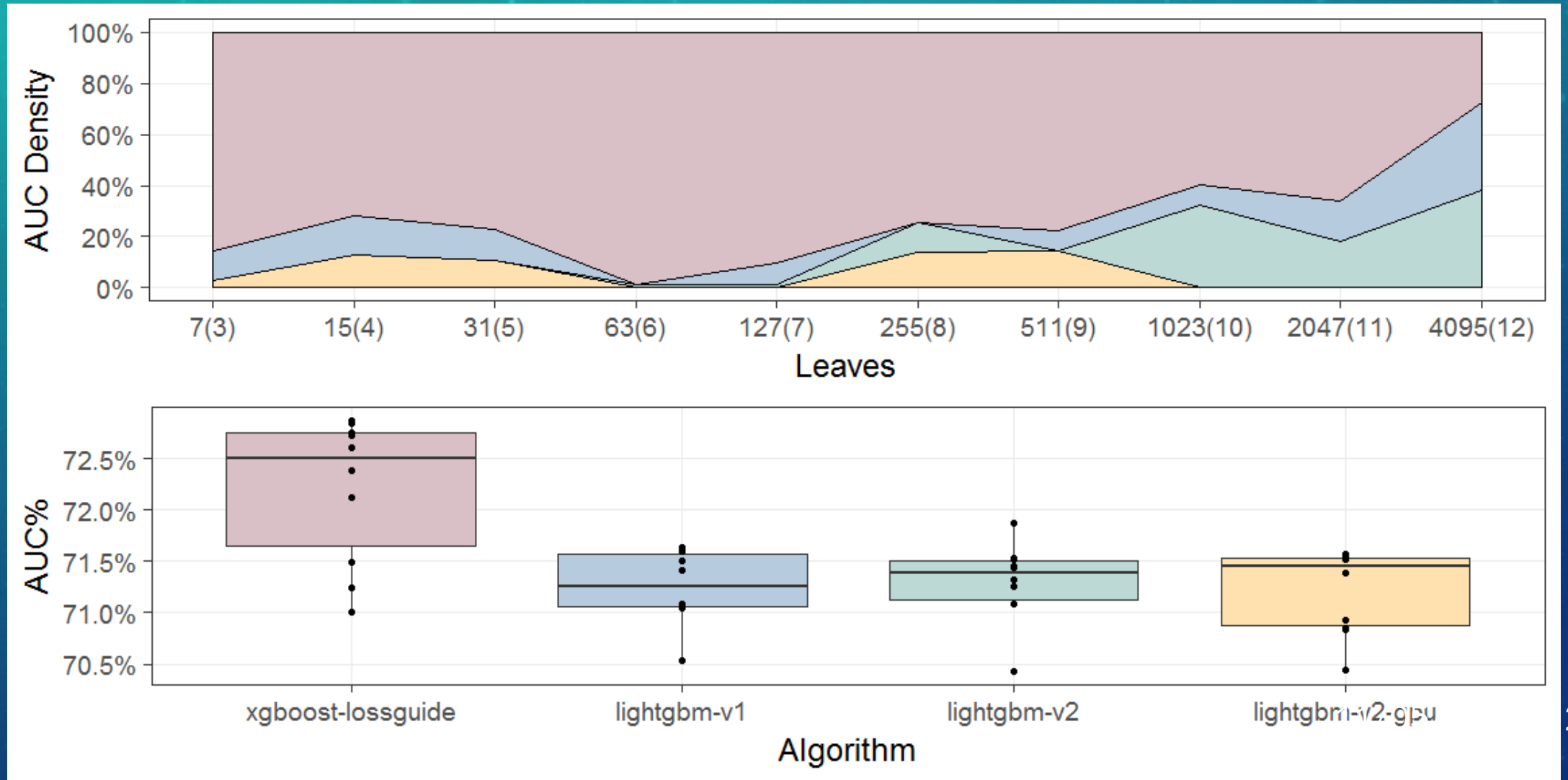
BOSCH PERFORMANCE

AUC vs Leaves – Line



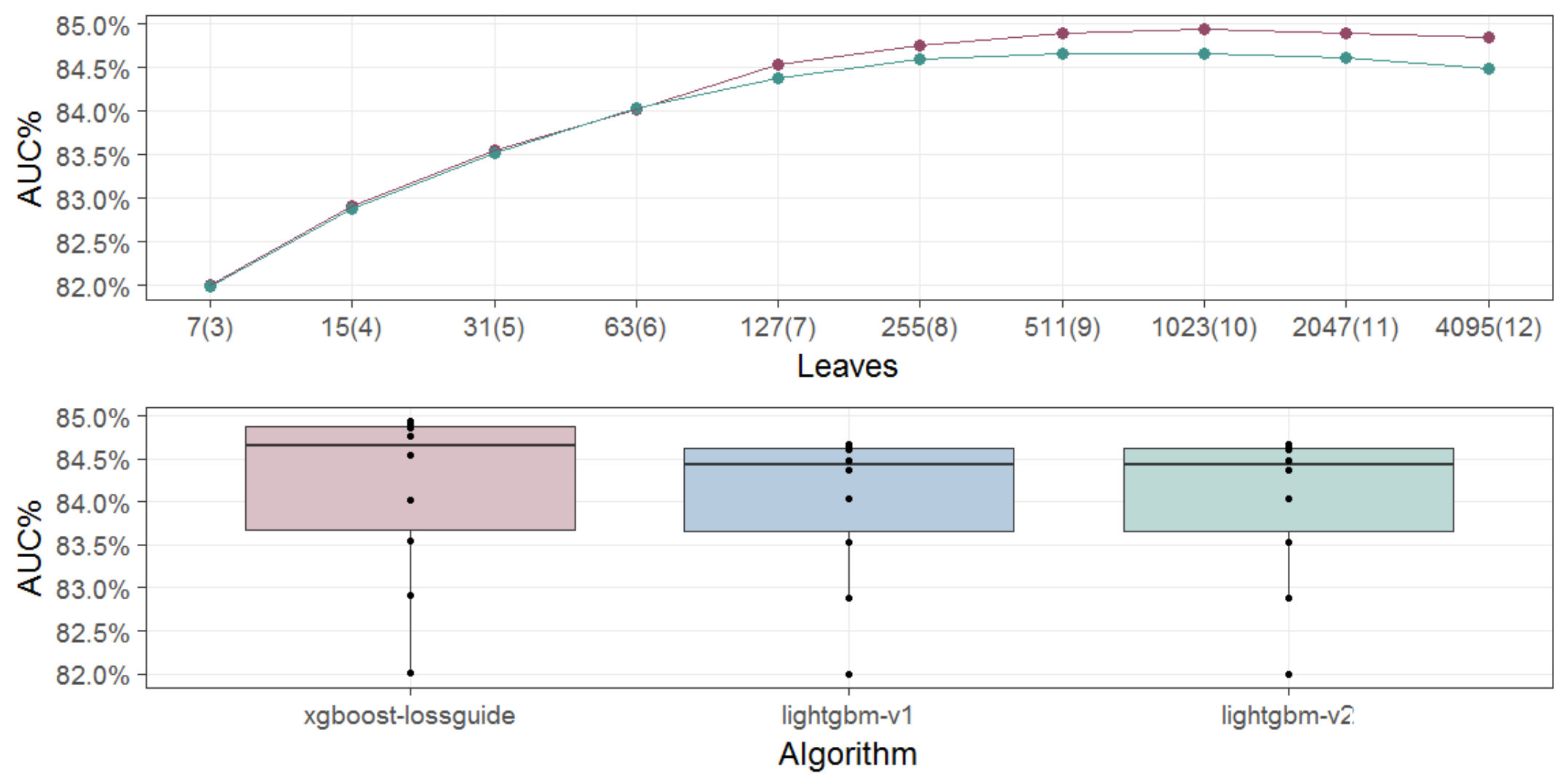
BOSCH PERFORMANCE

AUC vs Leaves – Density



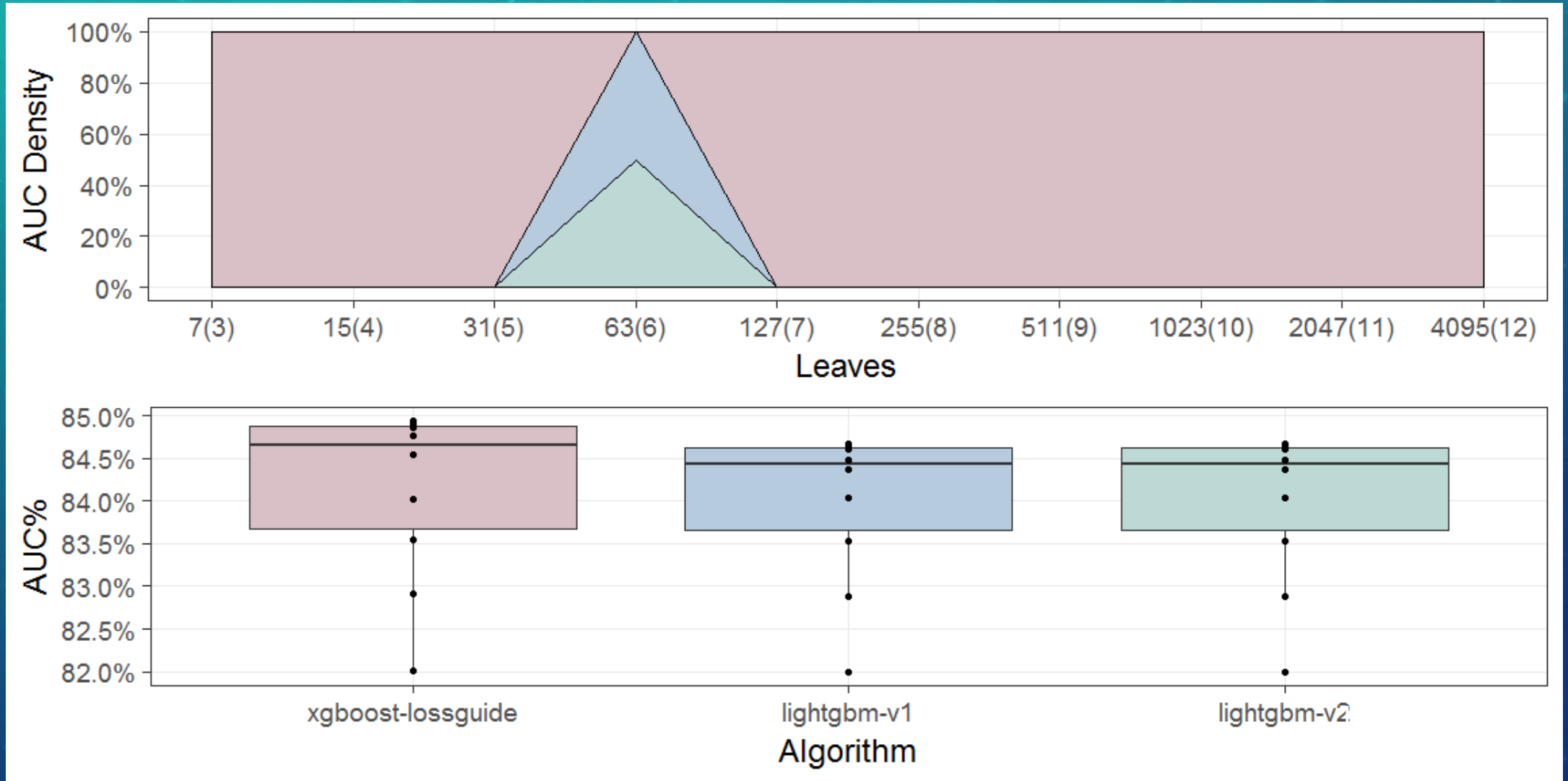
HIGGS PERFORMANCE

AUC vs Leaves – Line



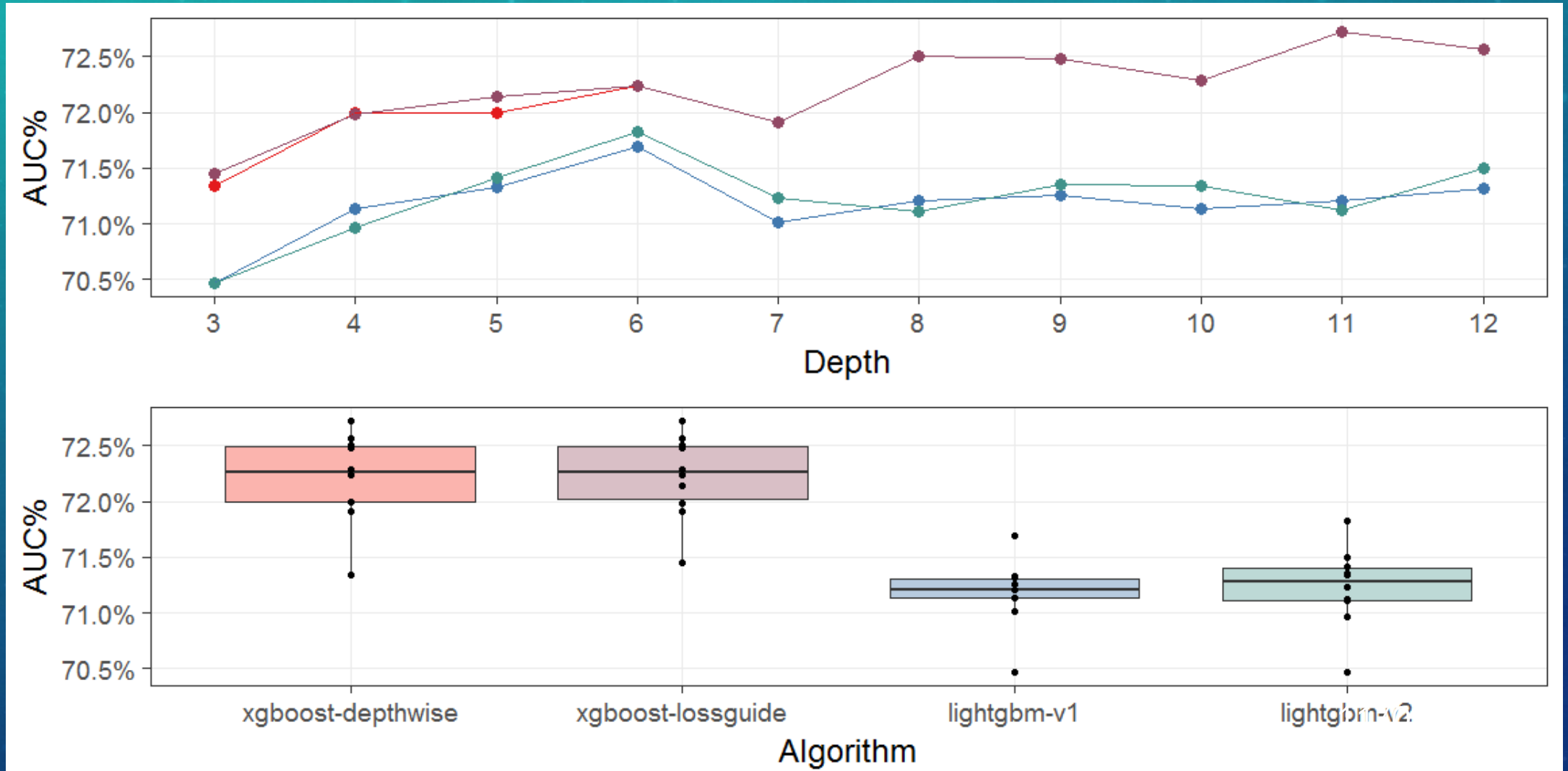
HIGGS PERFORMANCE

AUC vs Leaves – Density



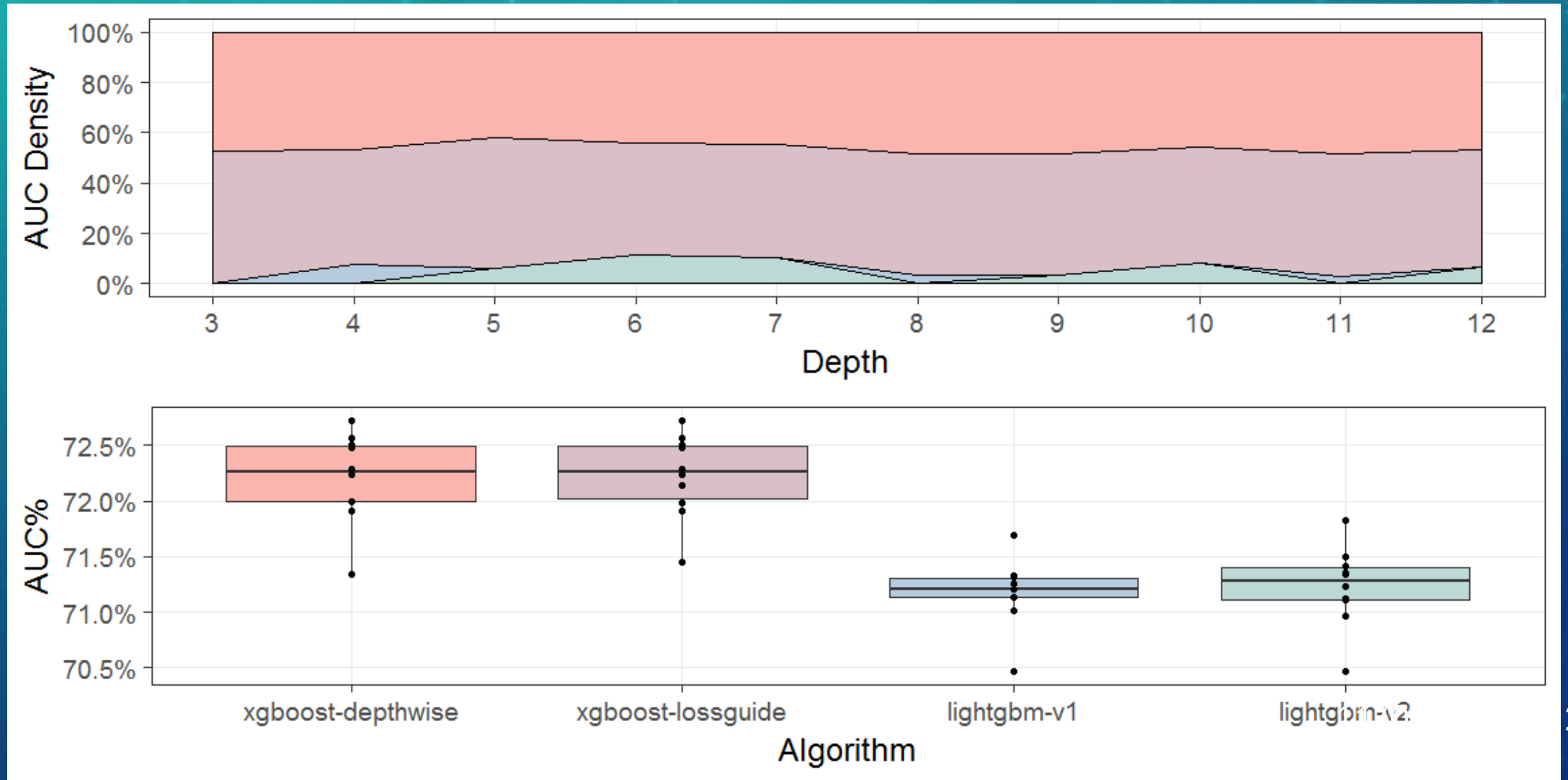
BOSCH PERFORMANCE

AUC vs Depth – Line



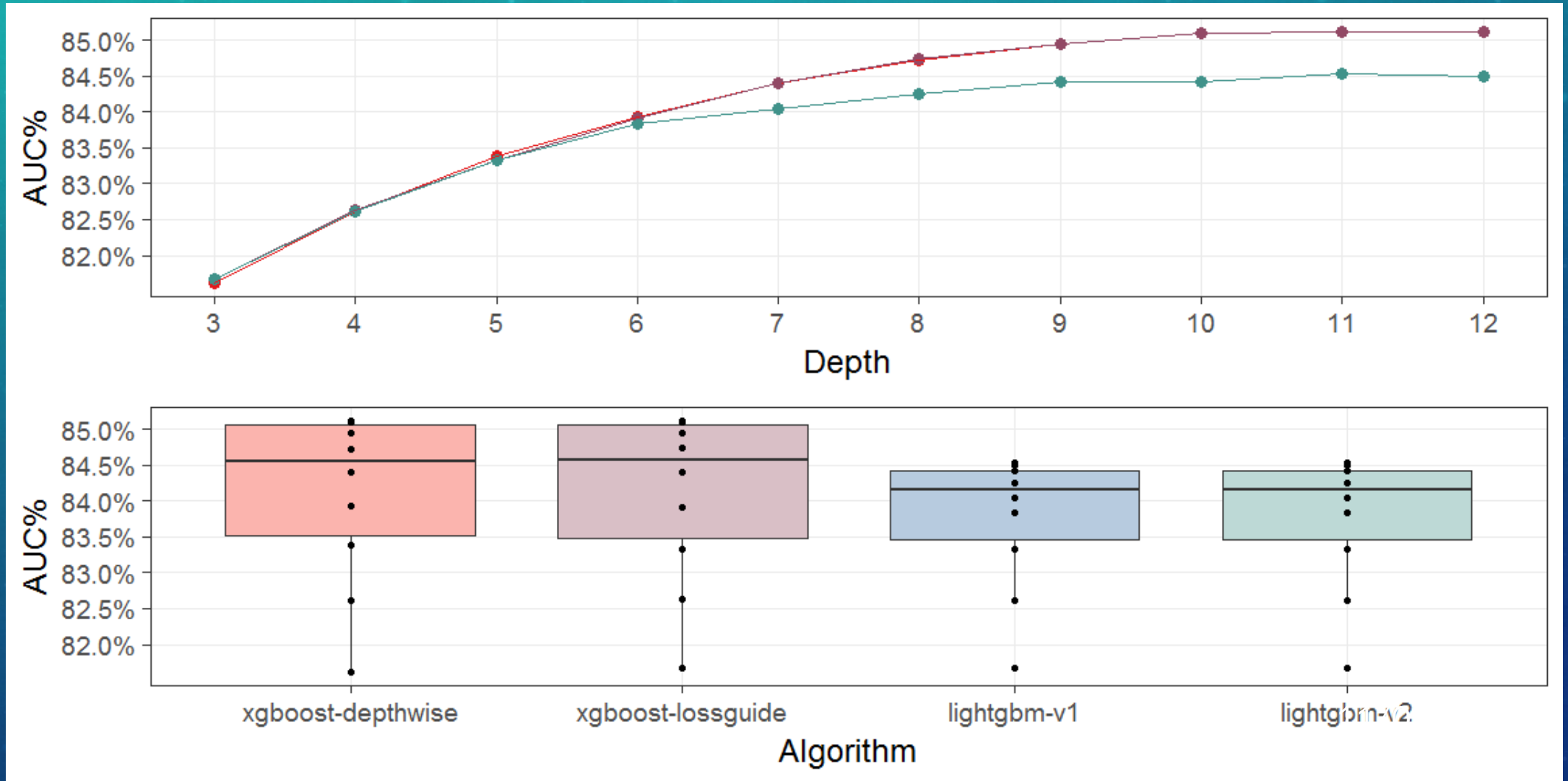
BOSCH PERFORMANCE

AUC vs Depth – Density



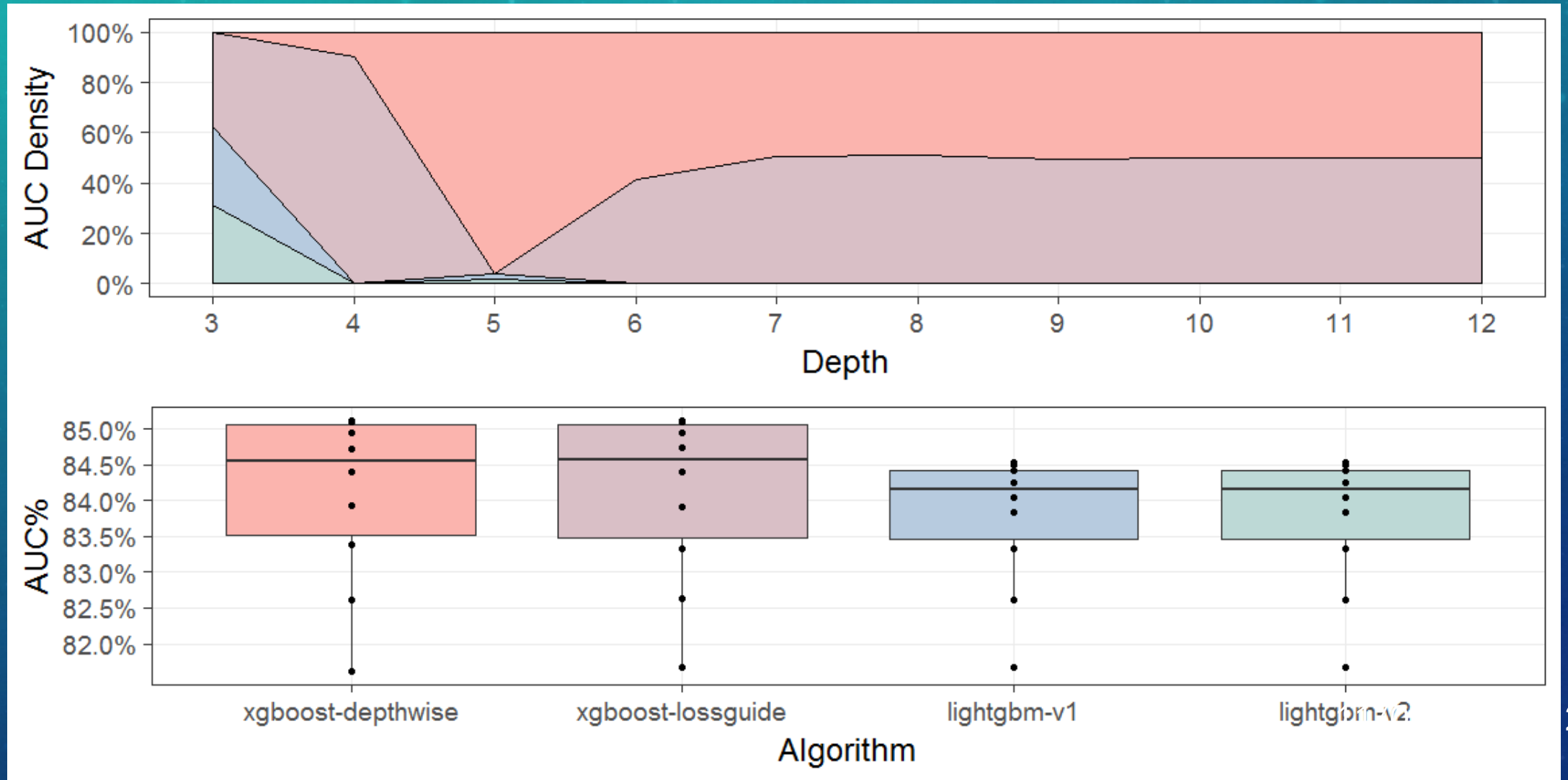
HIGGS PERFORMANCE

AUC vs Depth – Line



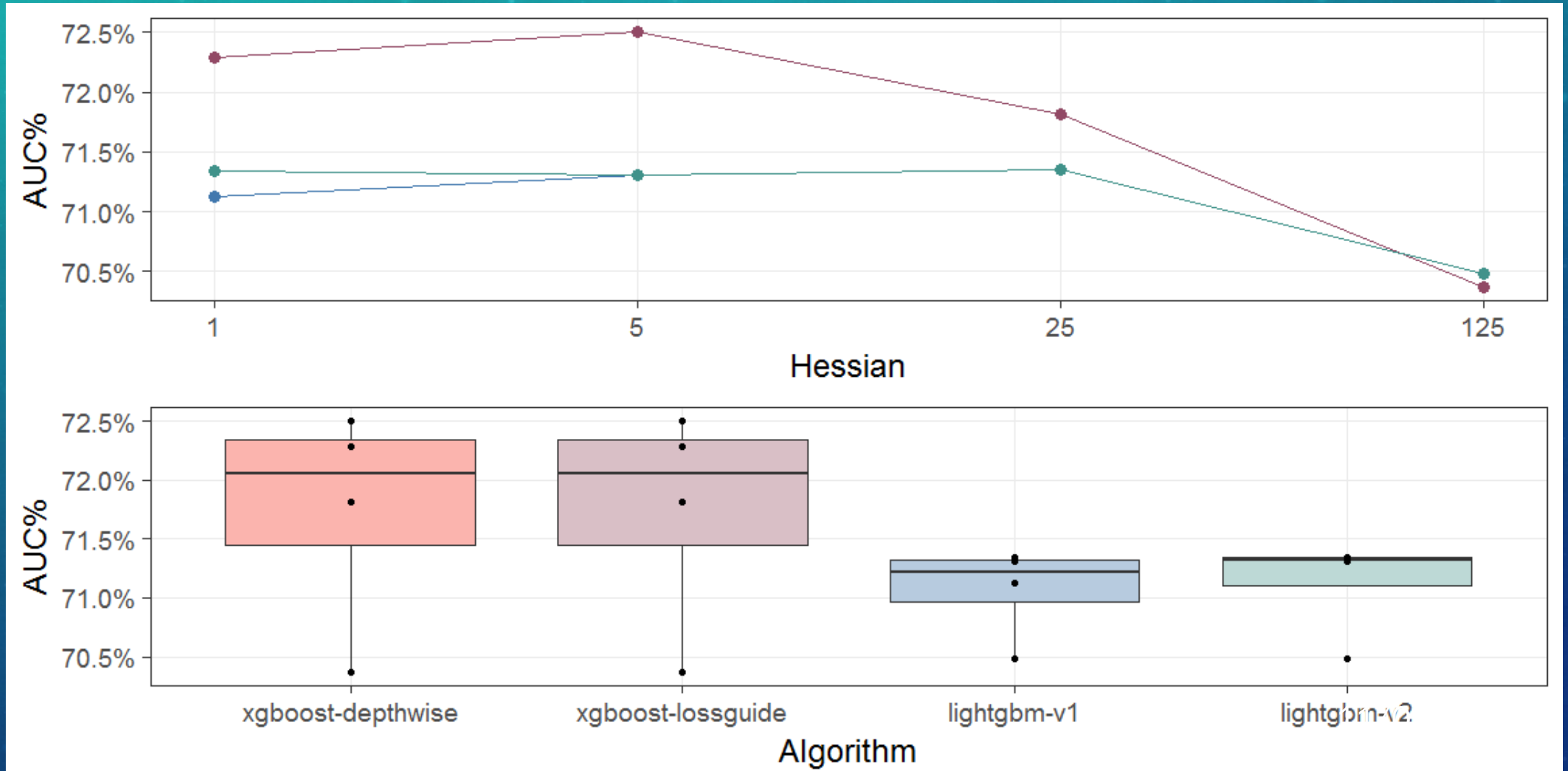
HIGGS PERFORMANCE

AUC vs Depth – Density



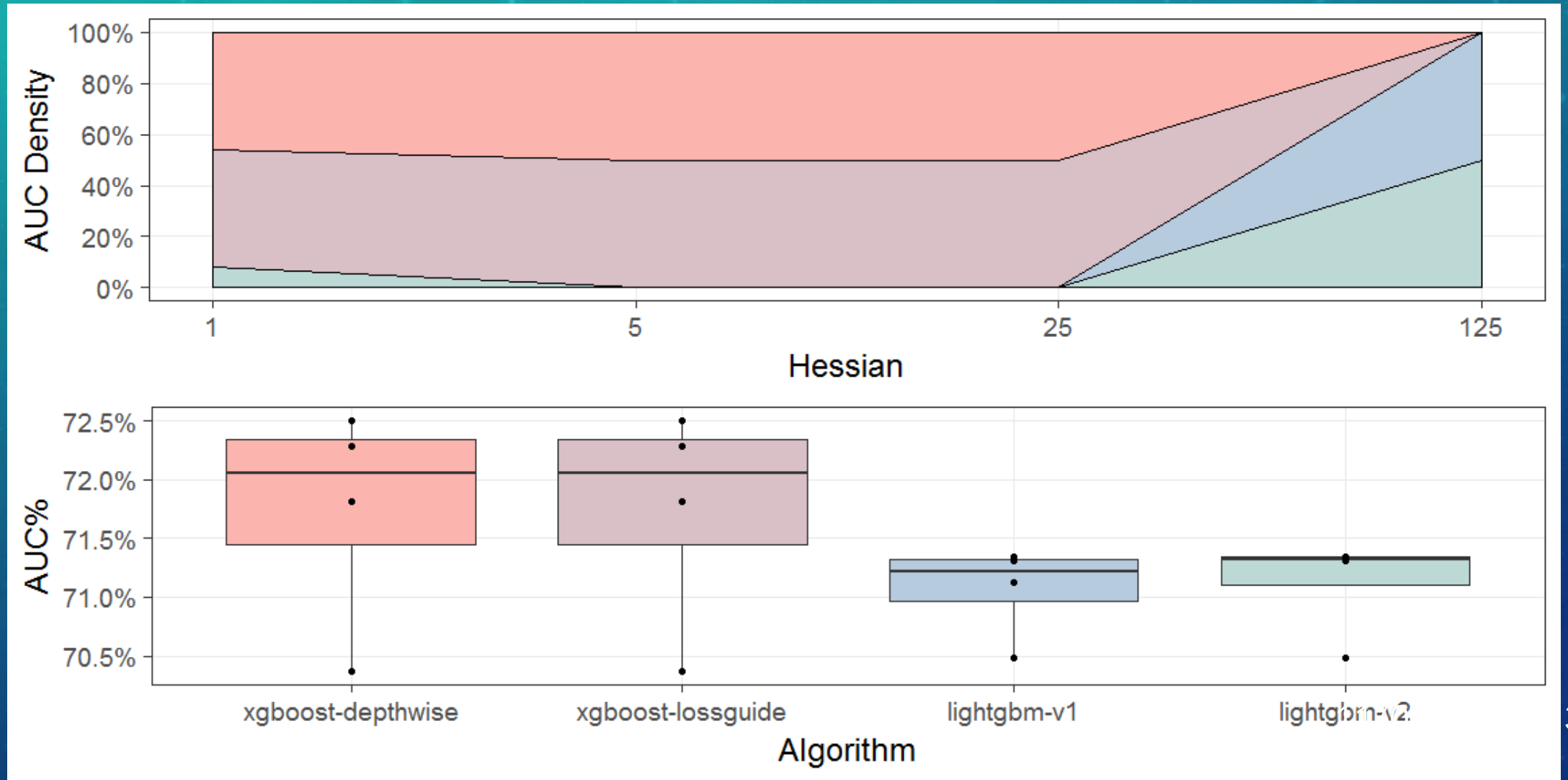
BOSCH PERFORMANCE

AUC vs Pruning (Hessian) – Line



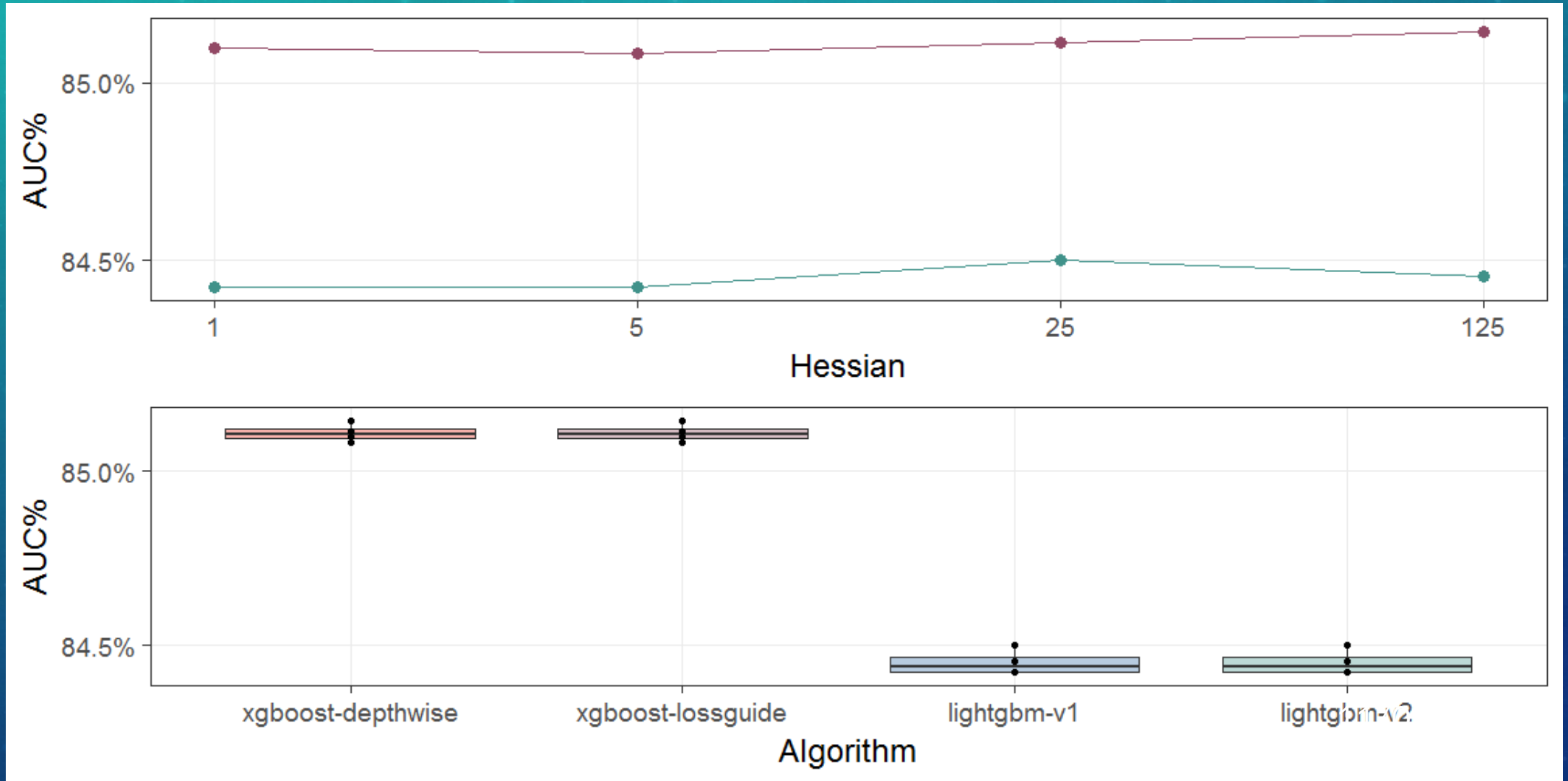
BOSCH PERFORMANCE

AUC vs Pruning (Hessian) – Density



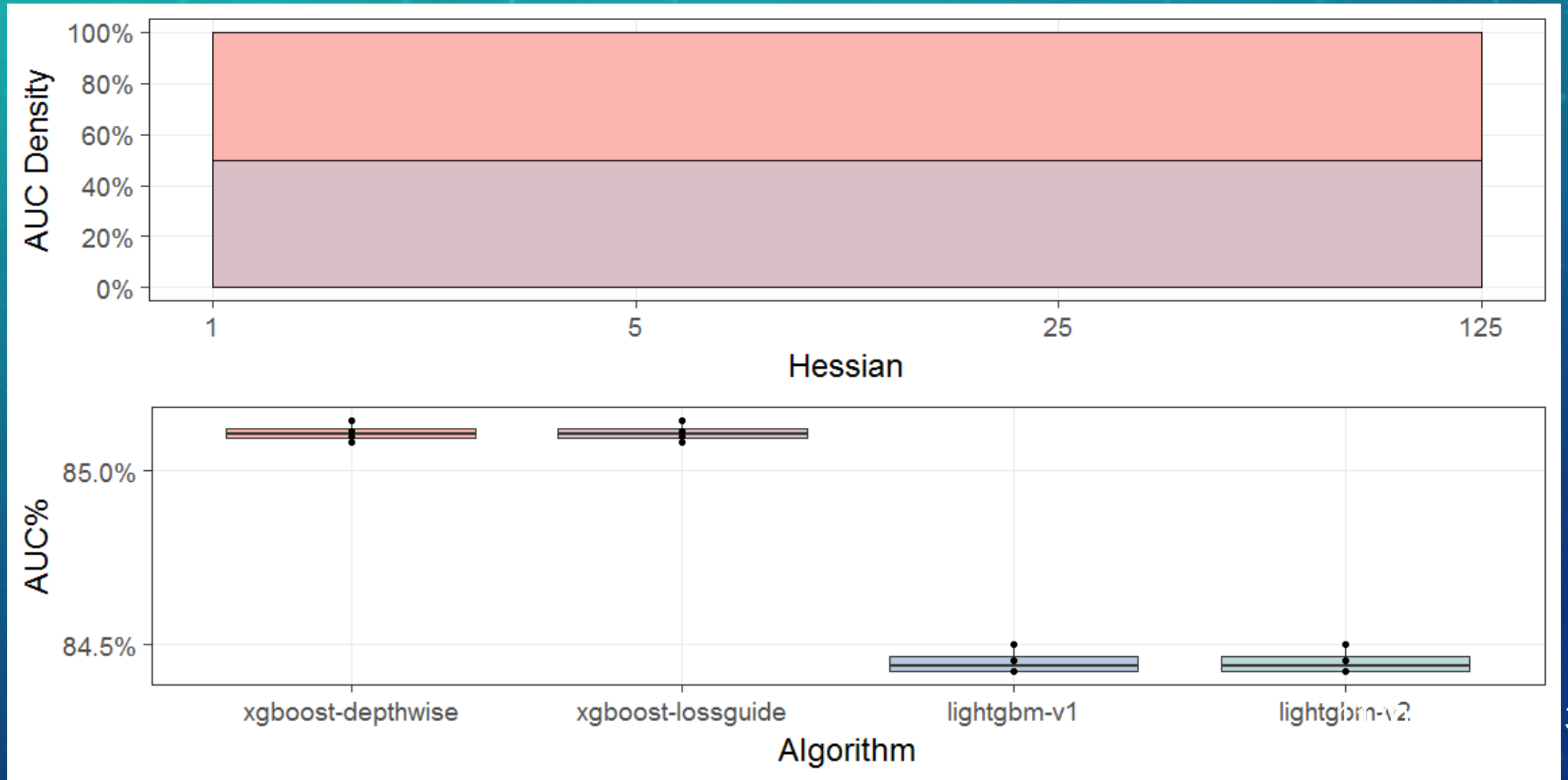
HIGGS PERFORMANCE

AUC vs Pruning (Hessian) – Line



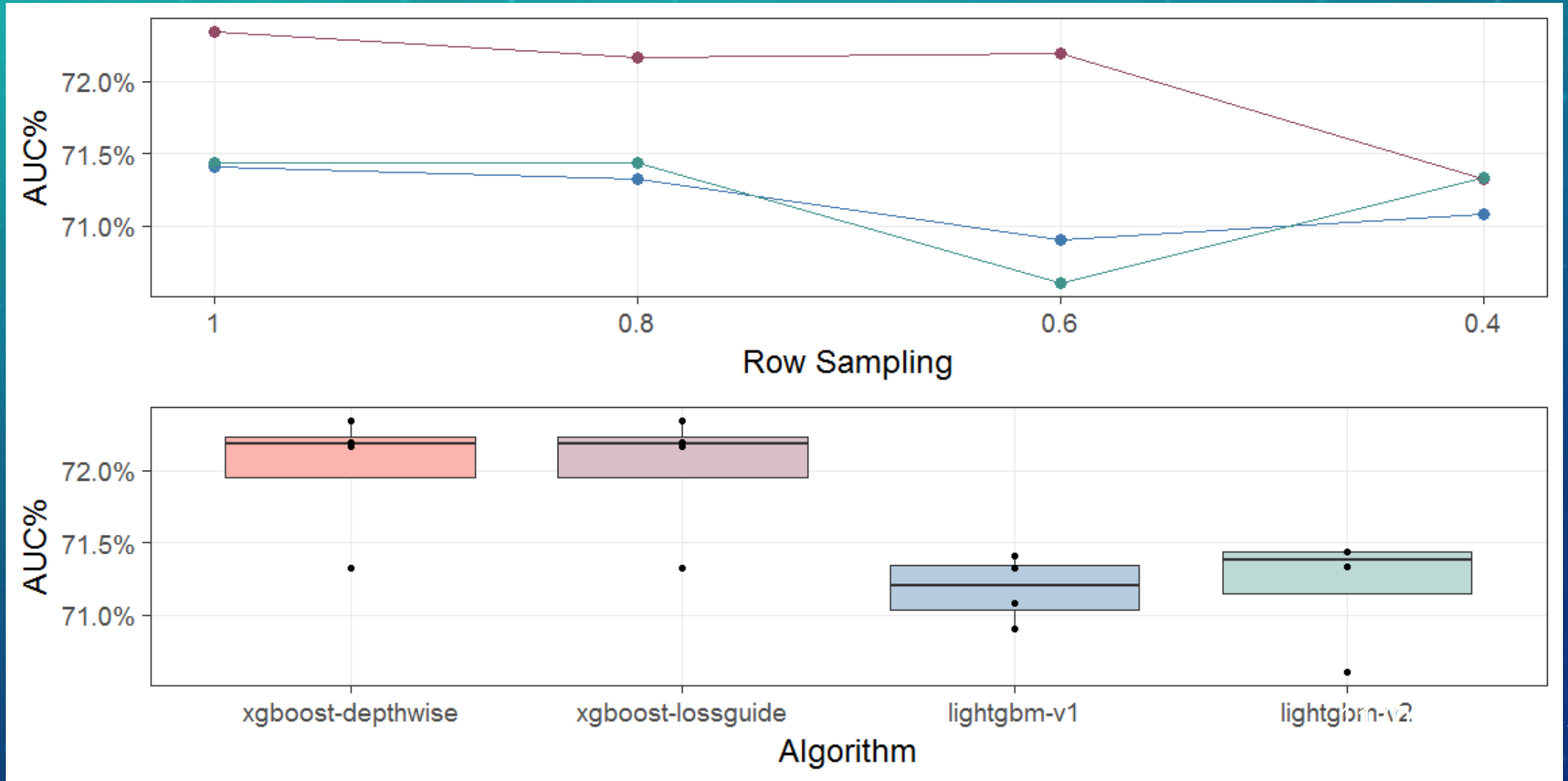
HIGGS PERFORMANCE

AUC vs Pruning (Hessian) – Density



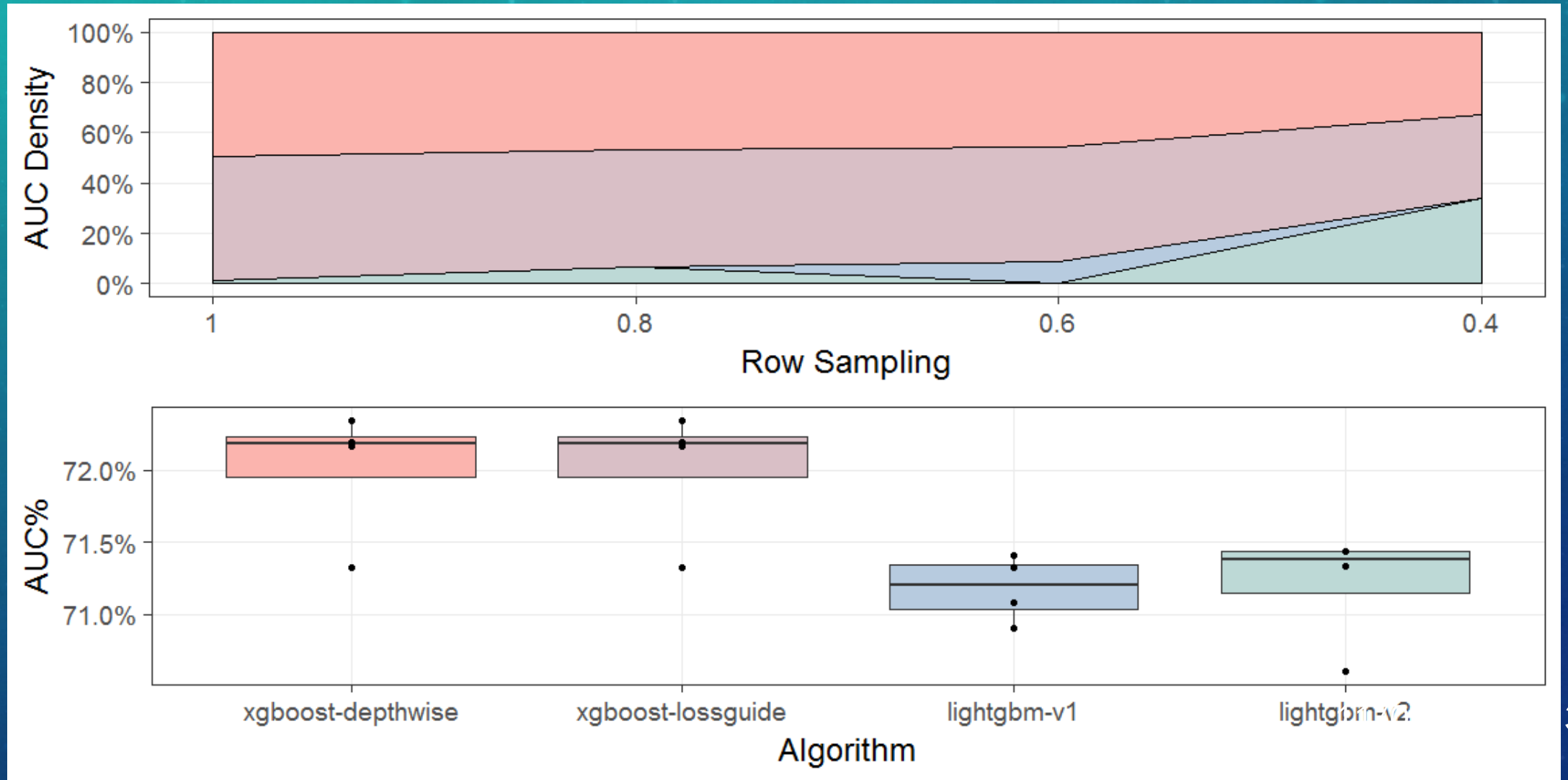
BOSCH PERFORMANCE

AUC vs Sampling (Stochastic) – Line



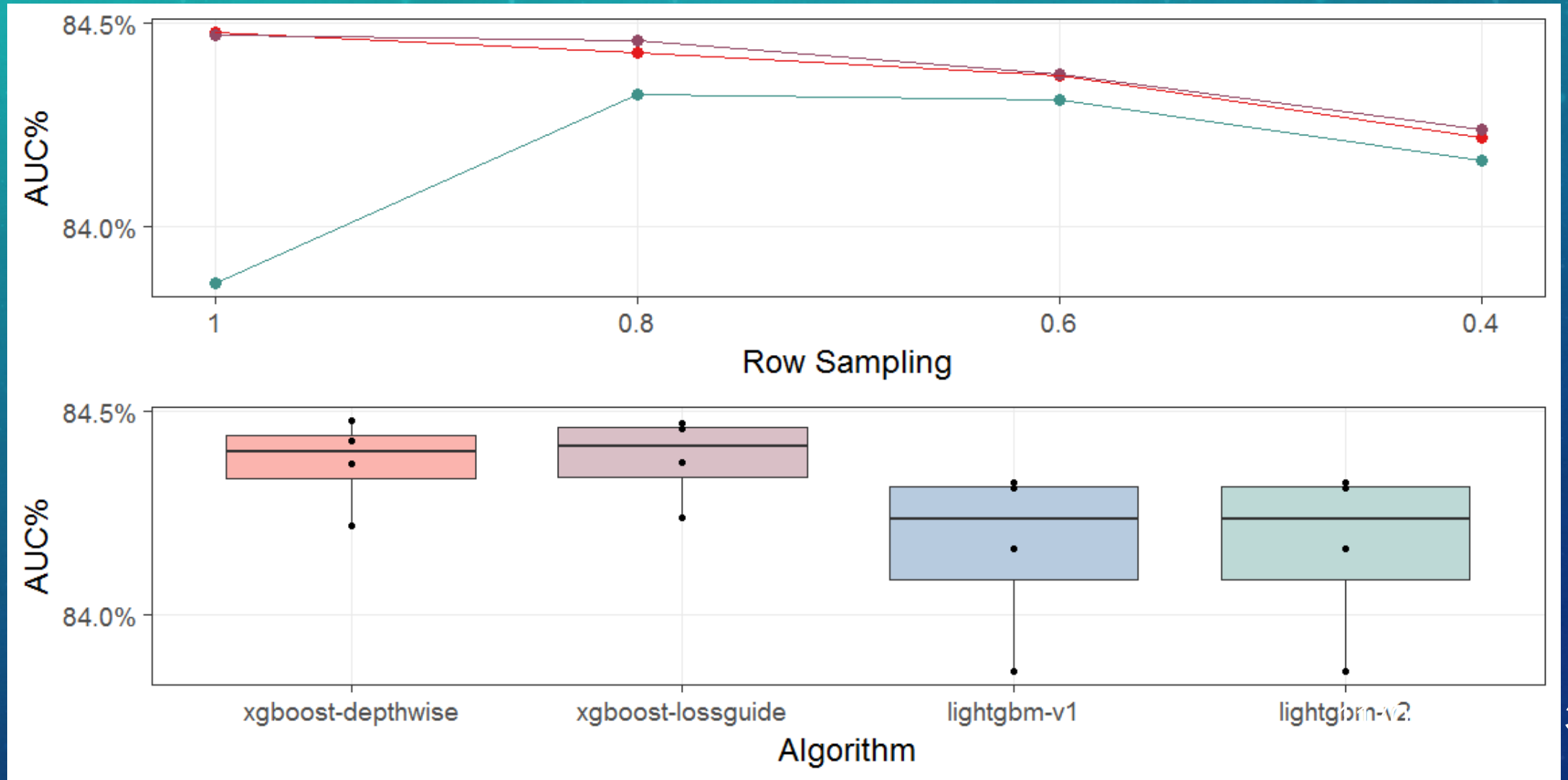
BOSCH PERFORMANCE

AUC vs Sampling (Stochastic) – Density



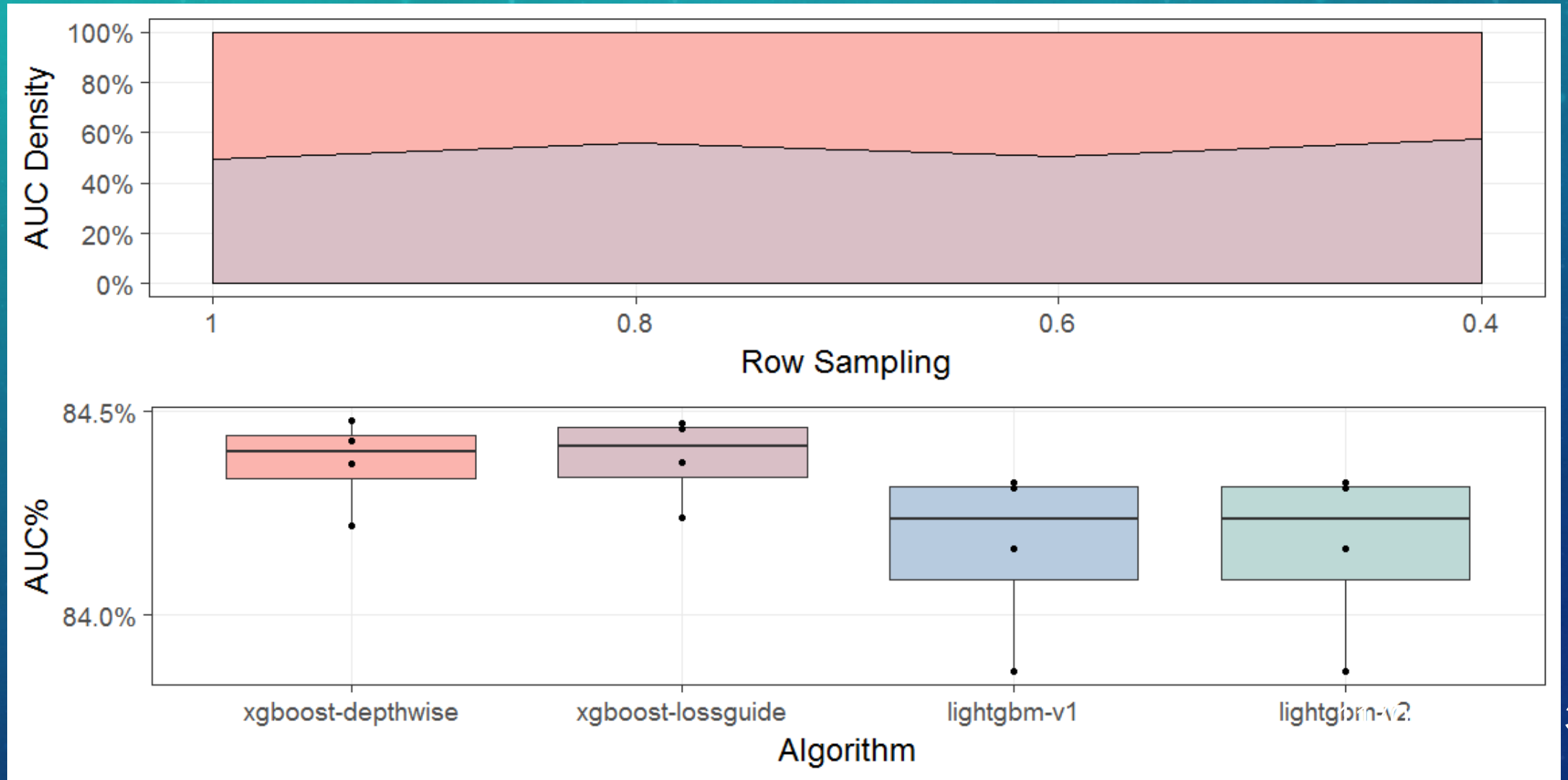
HIGGS PERFORMANCE

AUC vs Sampling (Stochastic) – Line



HIGGS PERFORMANCE

AUC vs Sampling (Stochastic) – Density



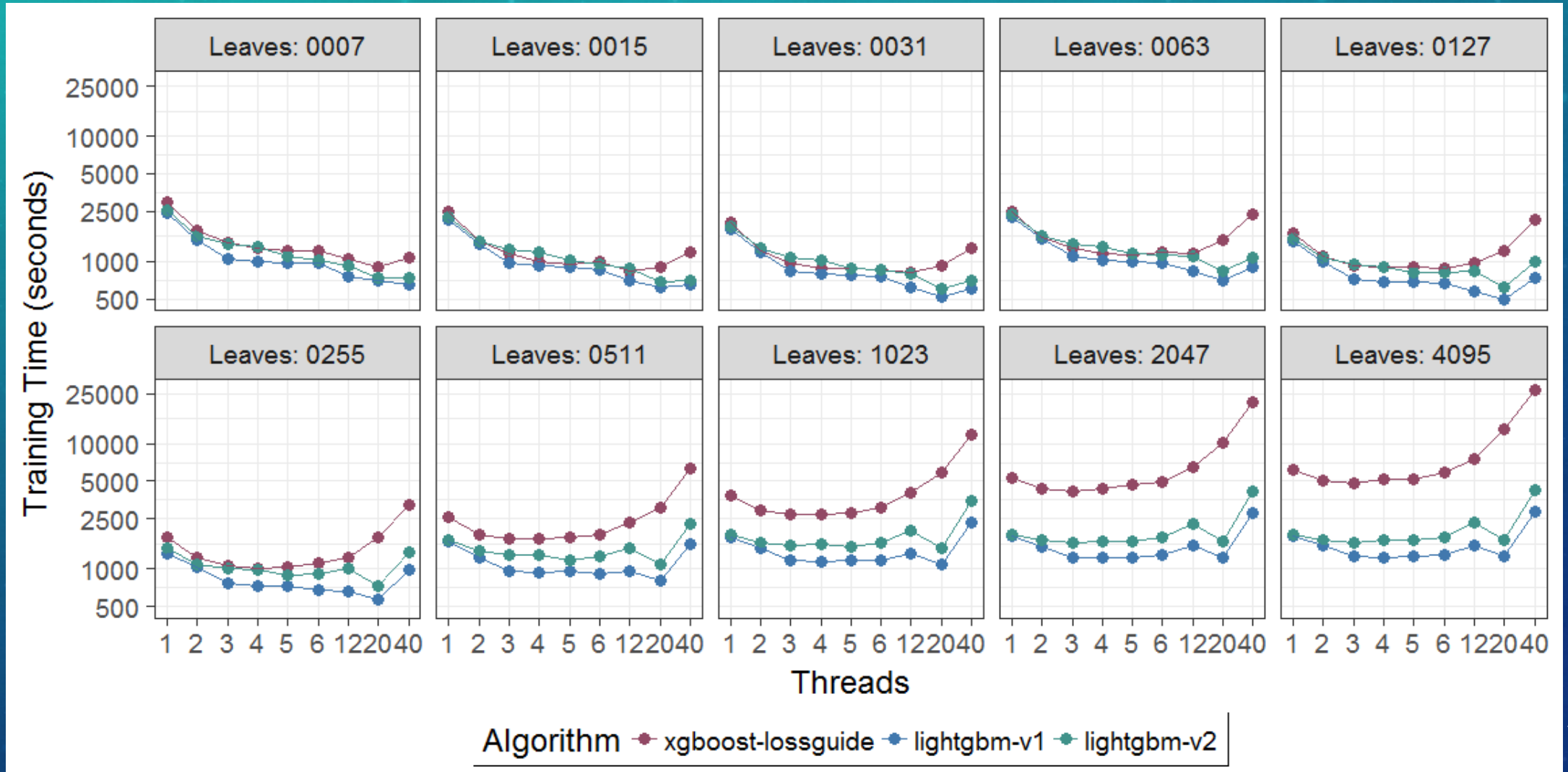
SPEED: HIGGS ISSUE?

Average of Best Iteration	Column Labels 'depth'									
Row Labels	1	2	3	4	5	6	7	8	9	10
'bosch'	1859.922481	1296.643411	977.4083333	827.525	376.1833333	338.175	342.5916667	326.8083333	373.8333333	323.5083333
'higgs'	500	500	500	500	424.1875	297.0625	260.5	221.8125	221.6875	202.0625
'lightgbm-v1'	500	500	500	500	383	317	268	201	183	153
'lightgbm-v2'	500	500	500	500	383	317	268	201	183	153
'xgboost-depthwise'	500	500	500	500	500	500	500	500	500	496
'xgboost-lossguide'	500	500	500	500	500	500	500	500	500	488

Algolia kaggle

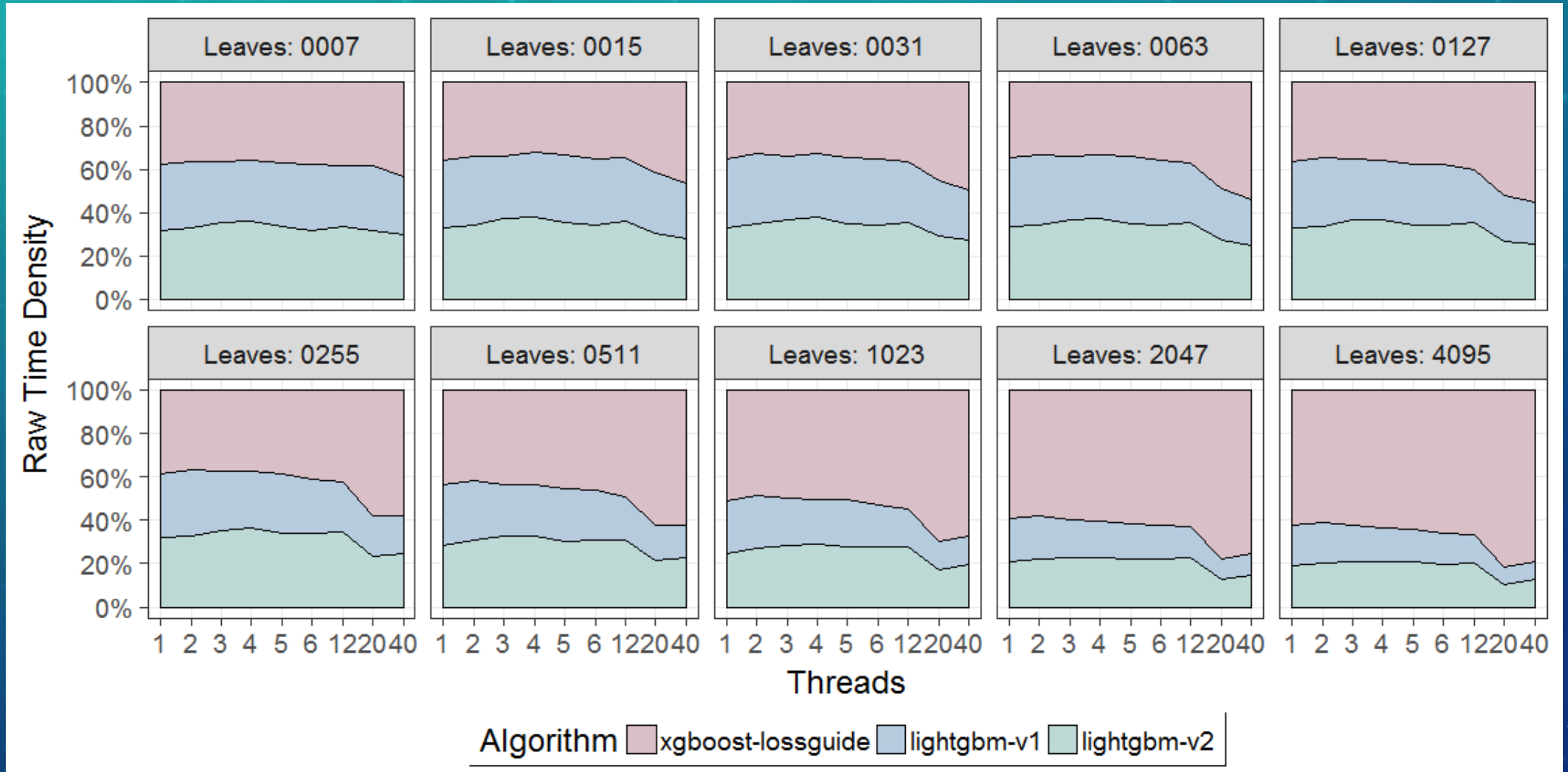
BOSCH SPEED

Time vs Leaves – Line



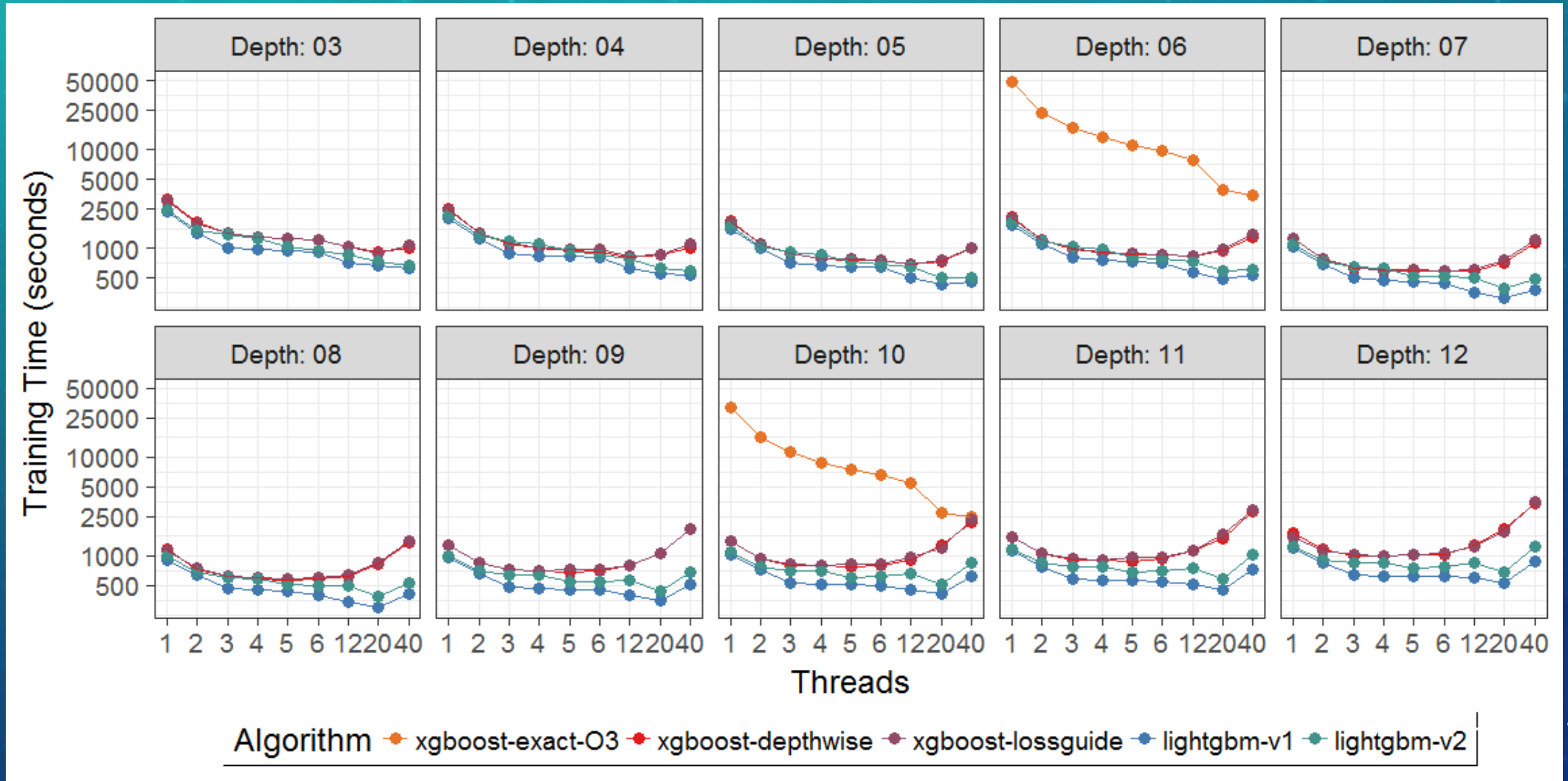
BOSCH SPEED

Time vs Leaves – Density



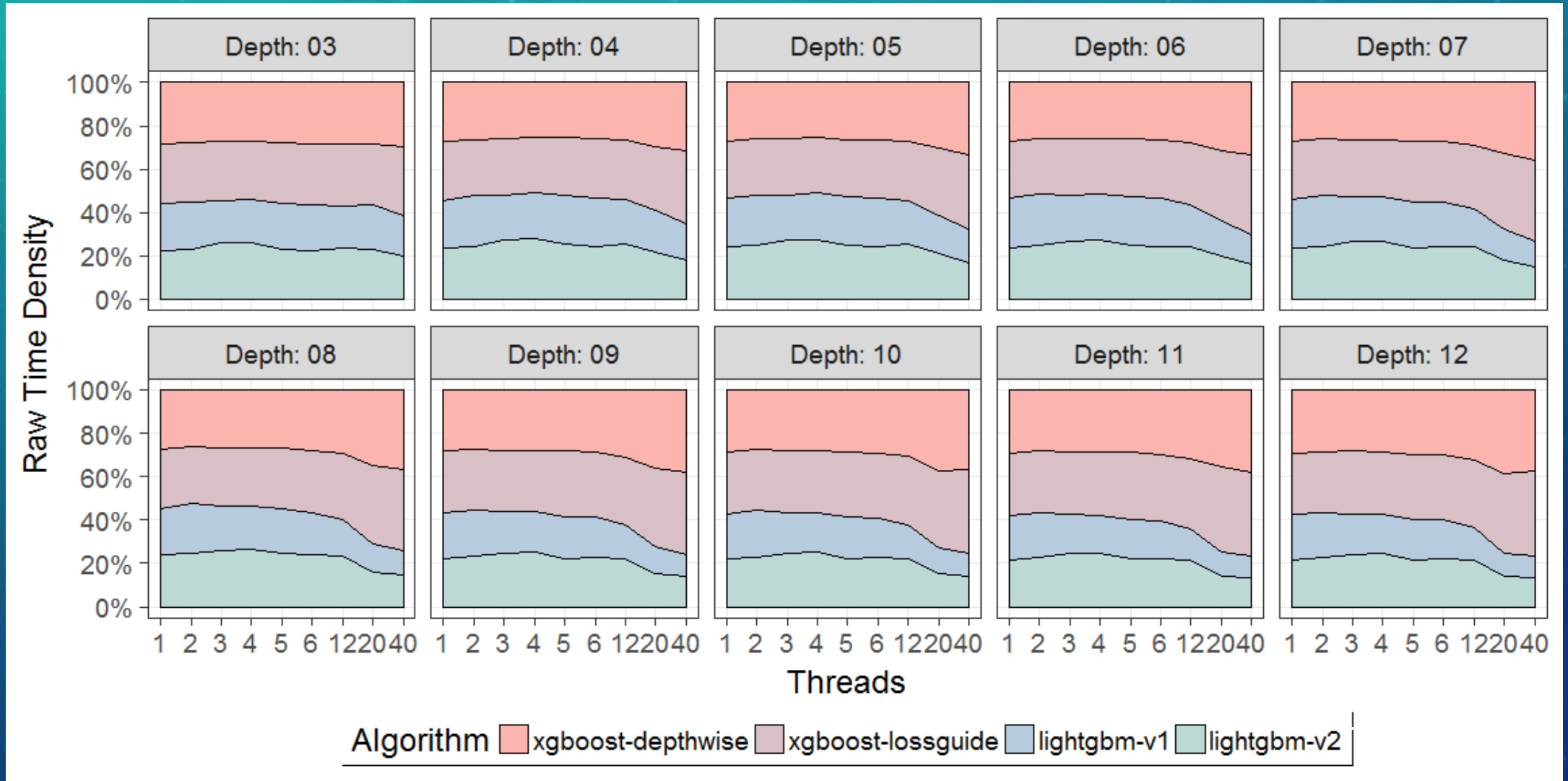
BOSCH SPEED

Time vs Depth – Line



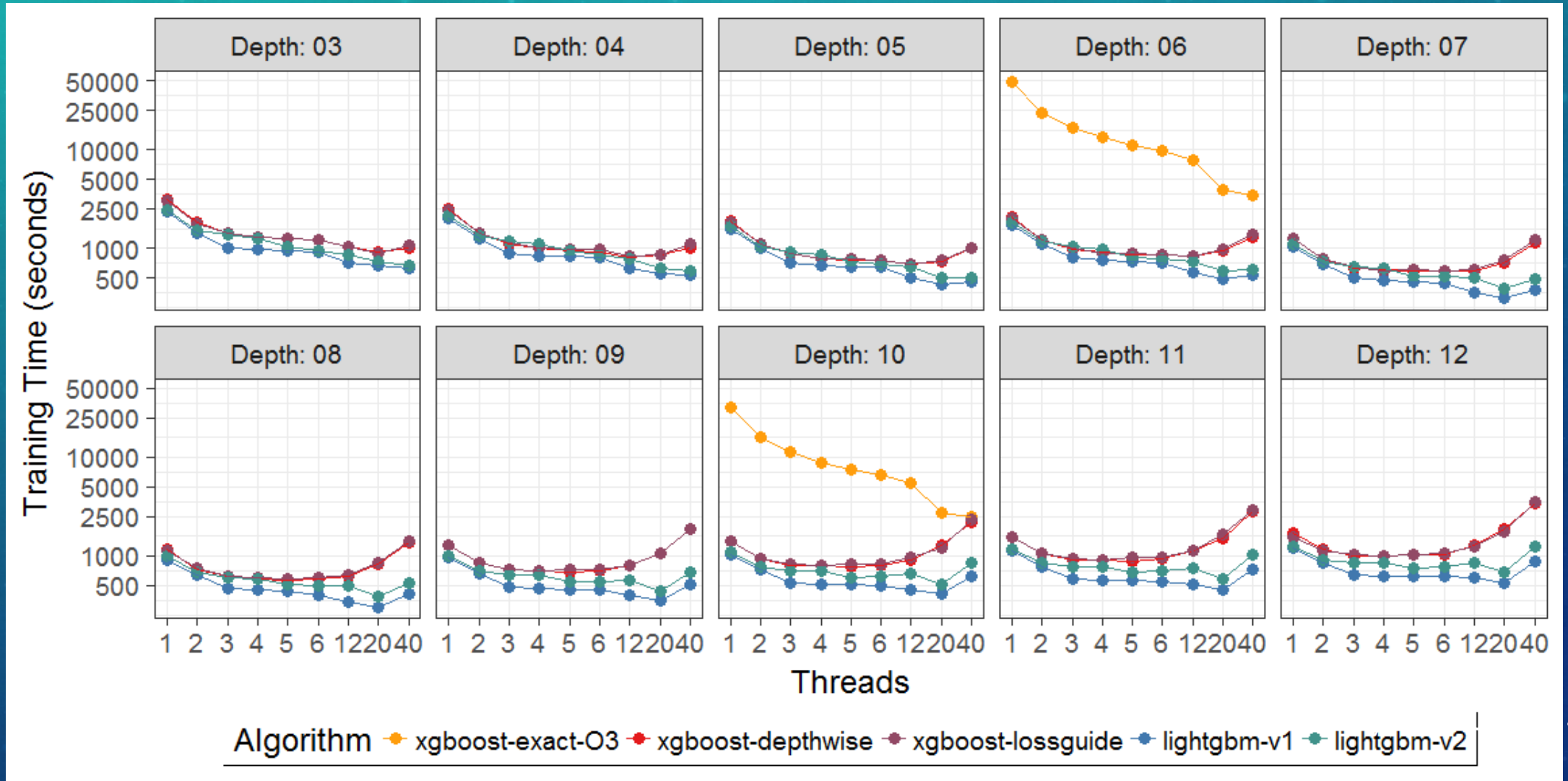
BOSCH SPEED

Time vs Depth – Density



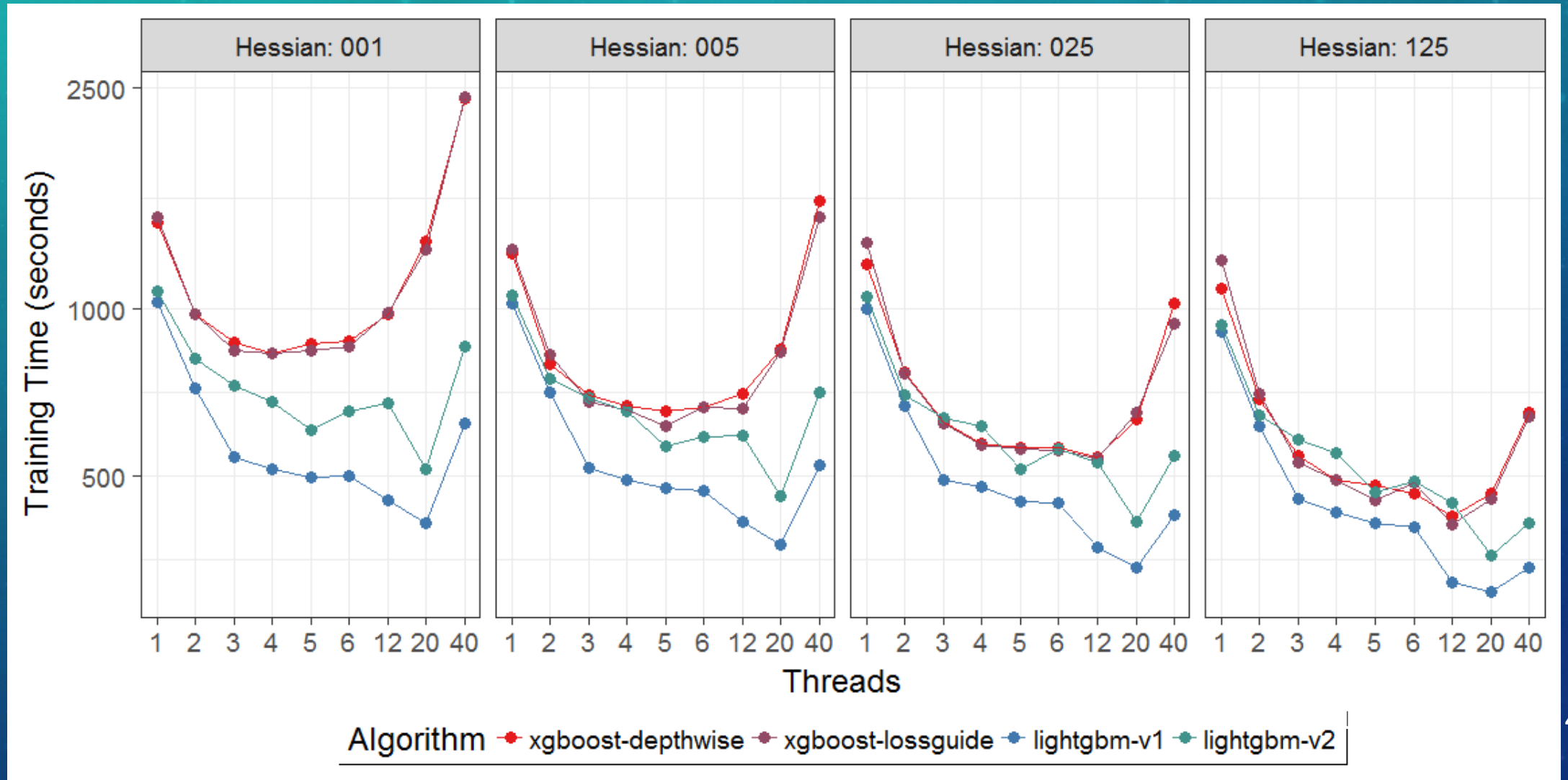
BOSCH SPEED

Time vs Depth – Exact



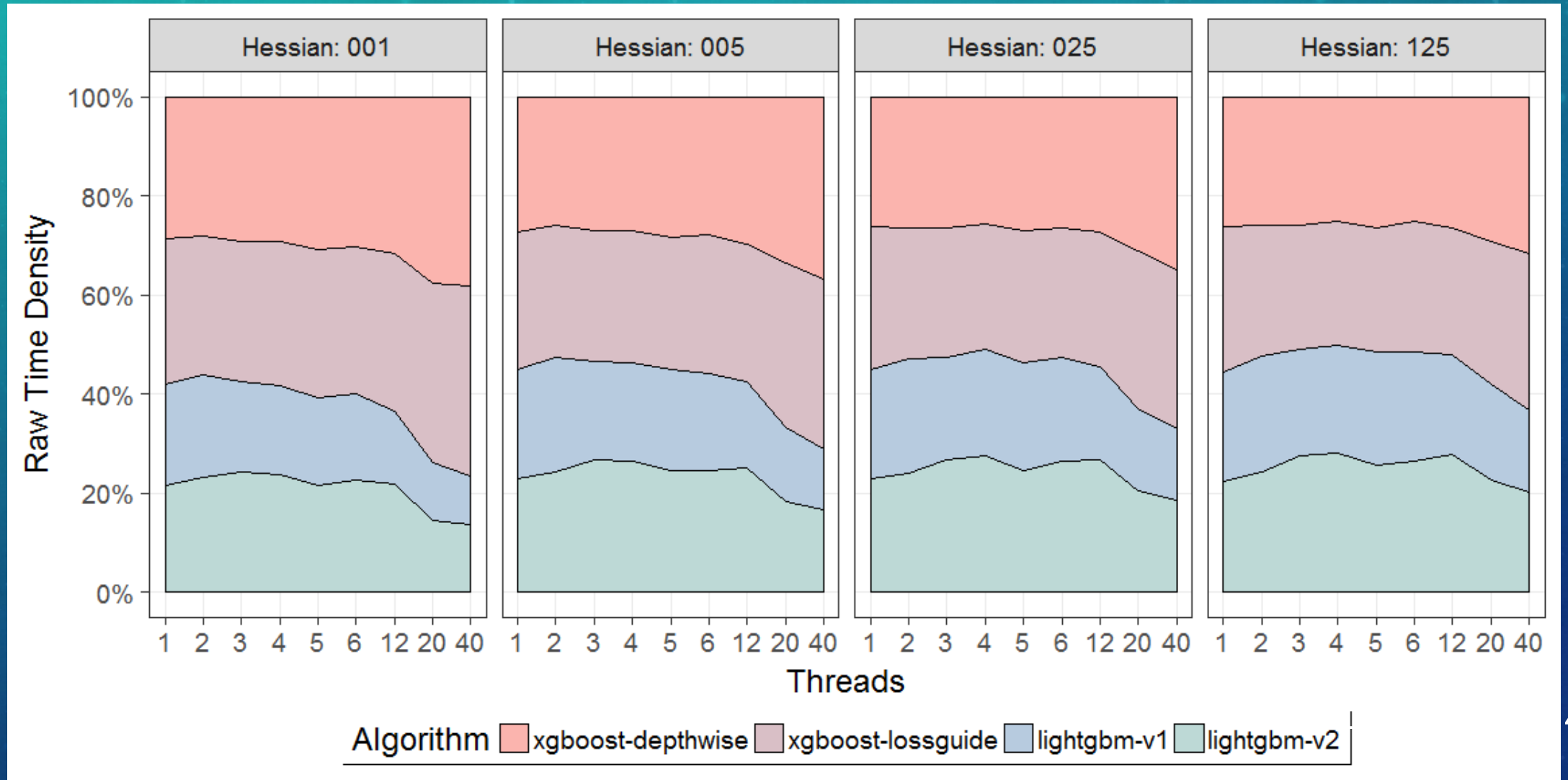
BOSCH SPEED

Time vs Pruning (Hessian) – Line



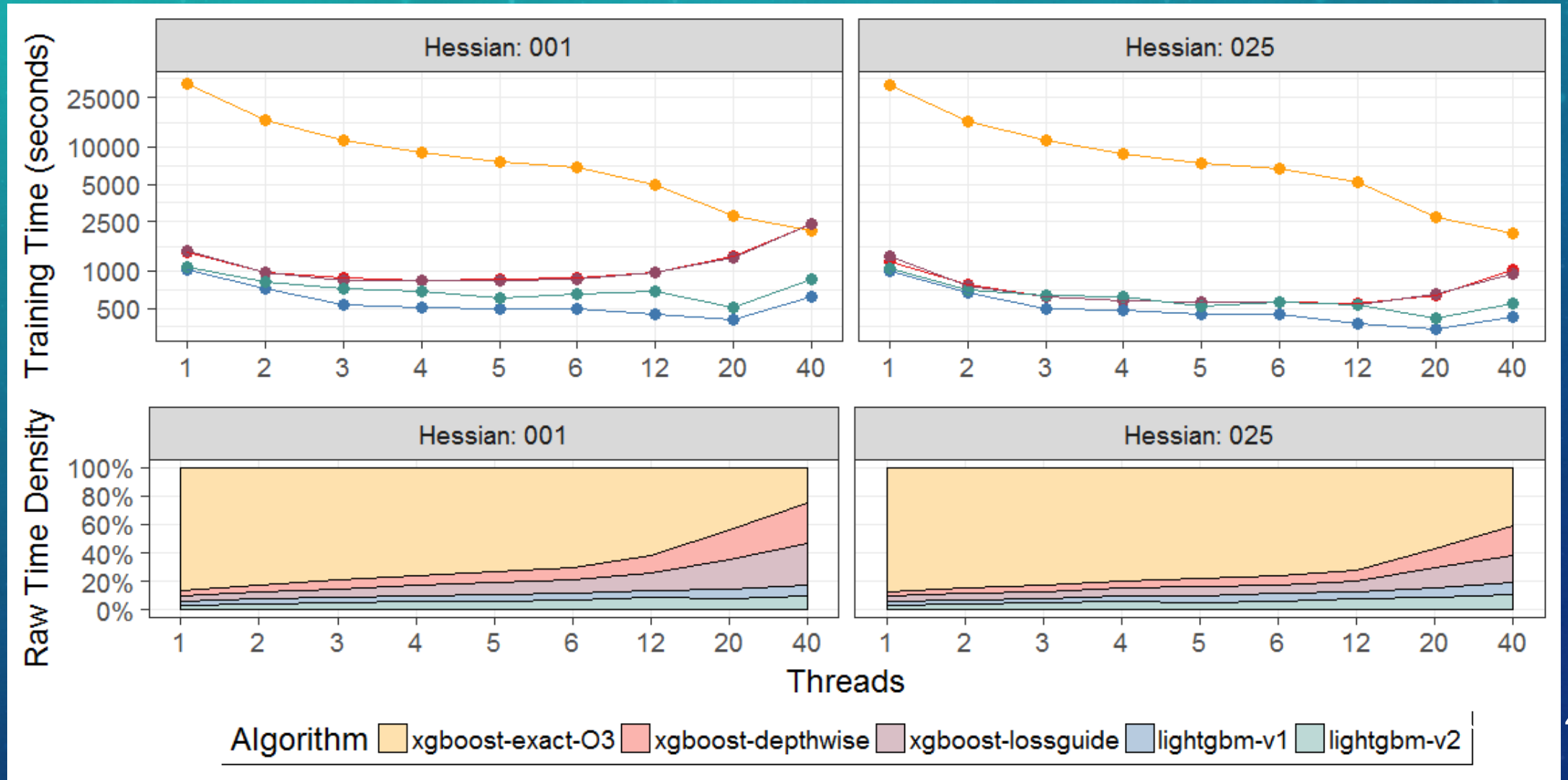
BOSCH SPEED

Time vs Pruning (Hessian) – Density



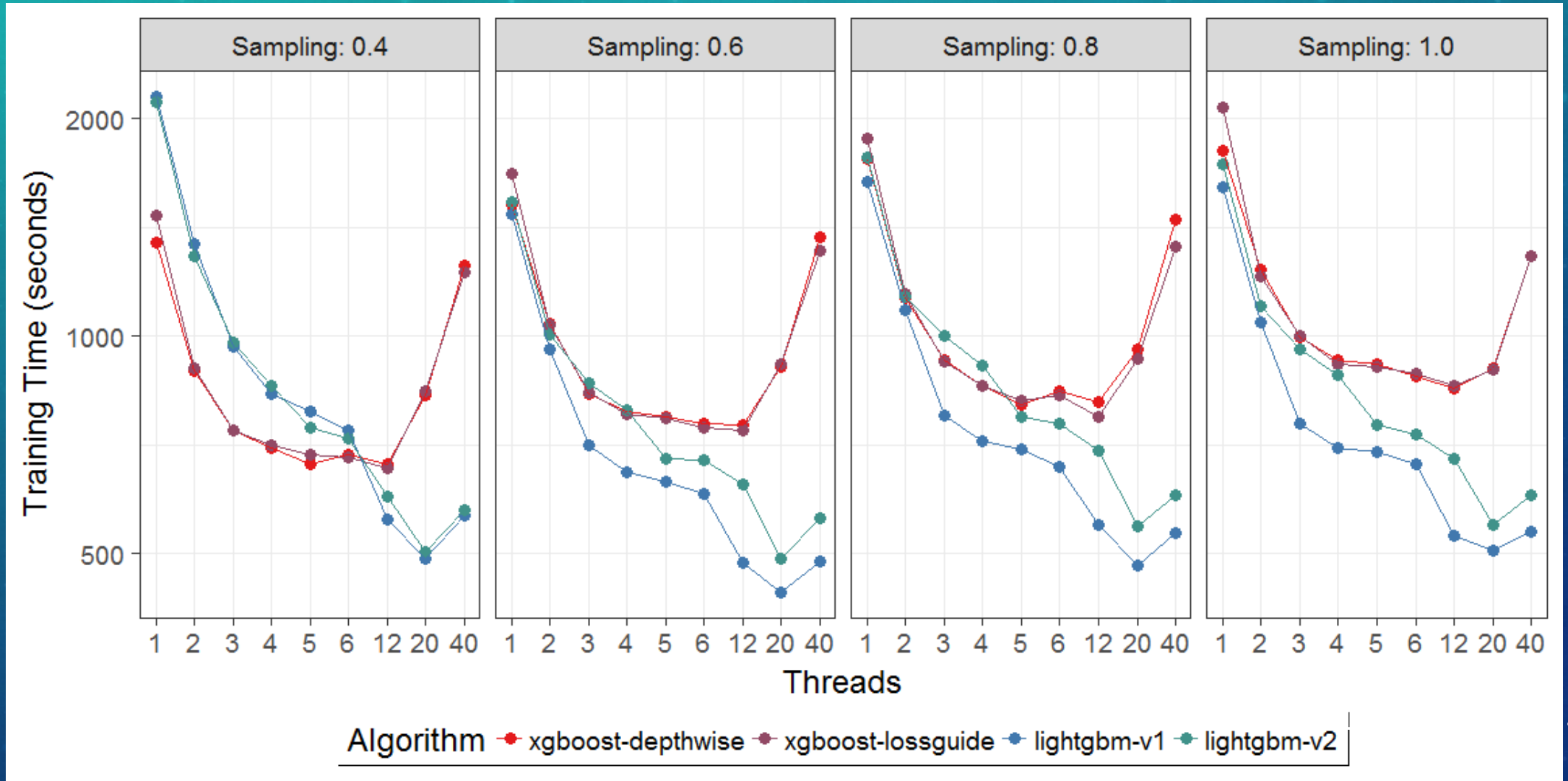
BOSCH SPEED

Time vs Pruning (Hessian) – Exact



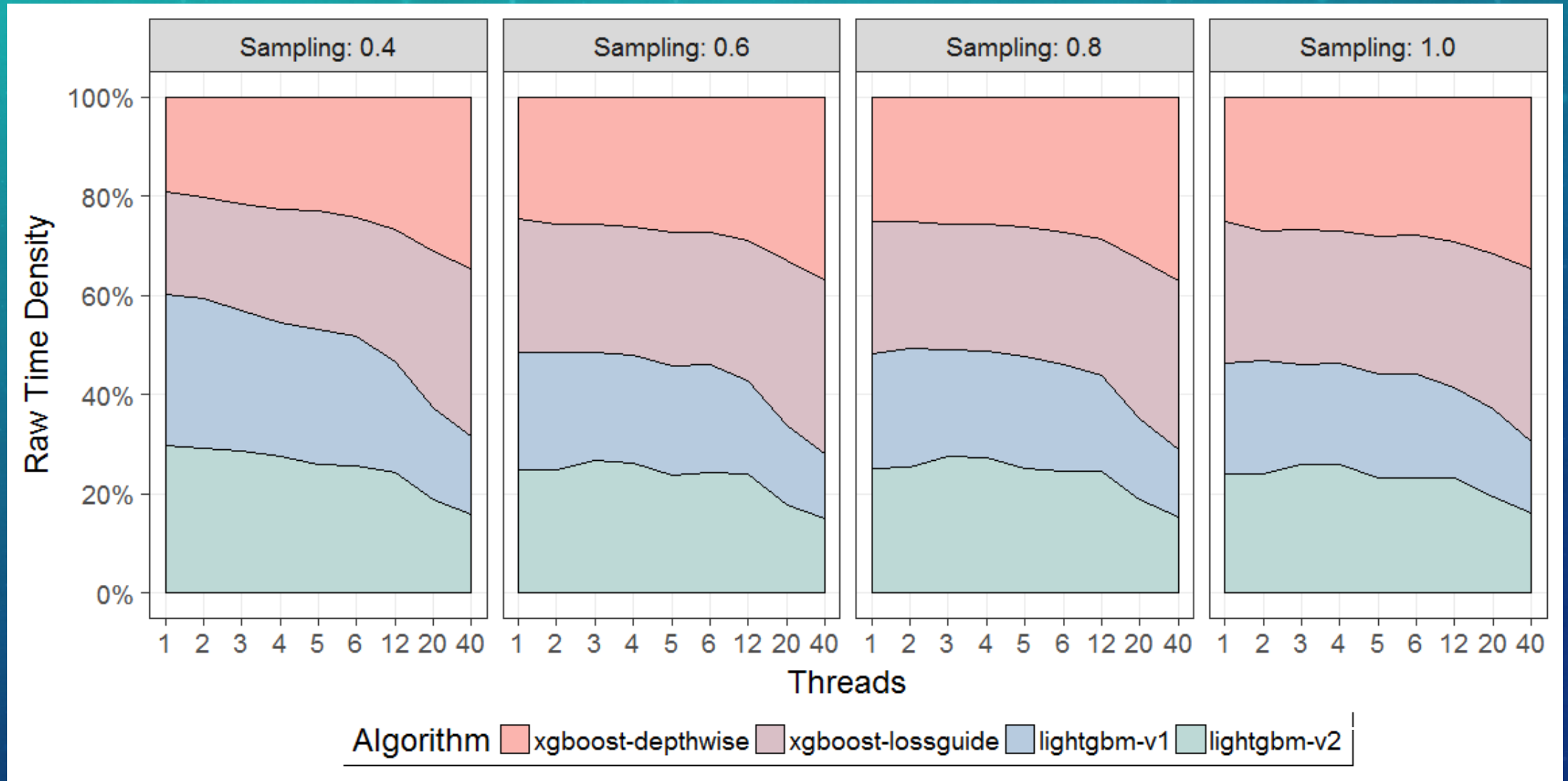
BOSCH SPEED

Time vs Sampling (Stochastic) – Line



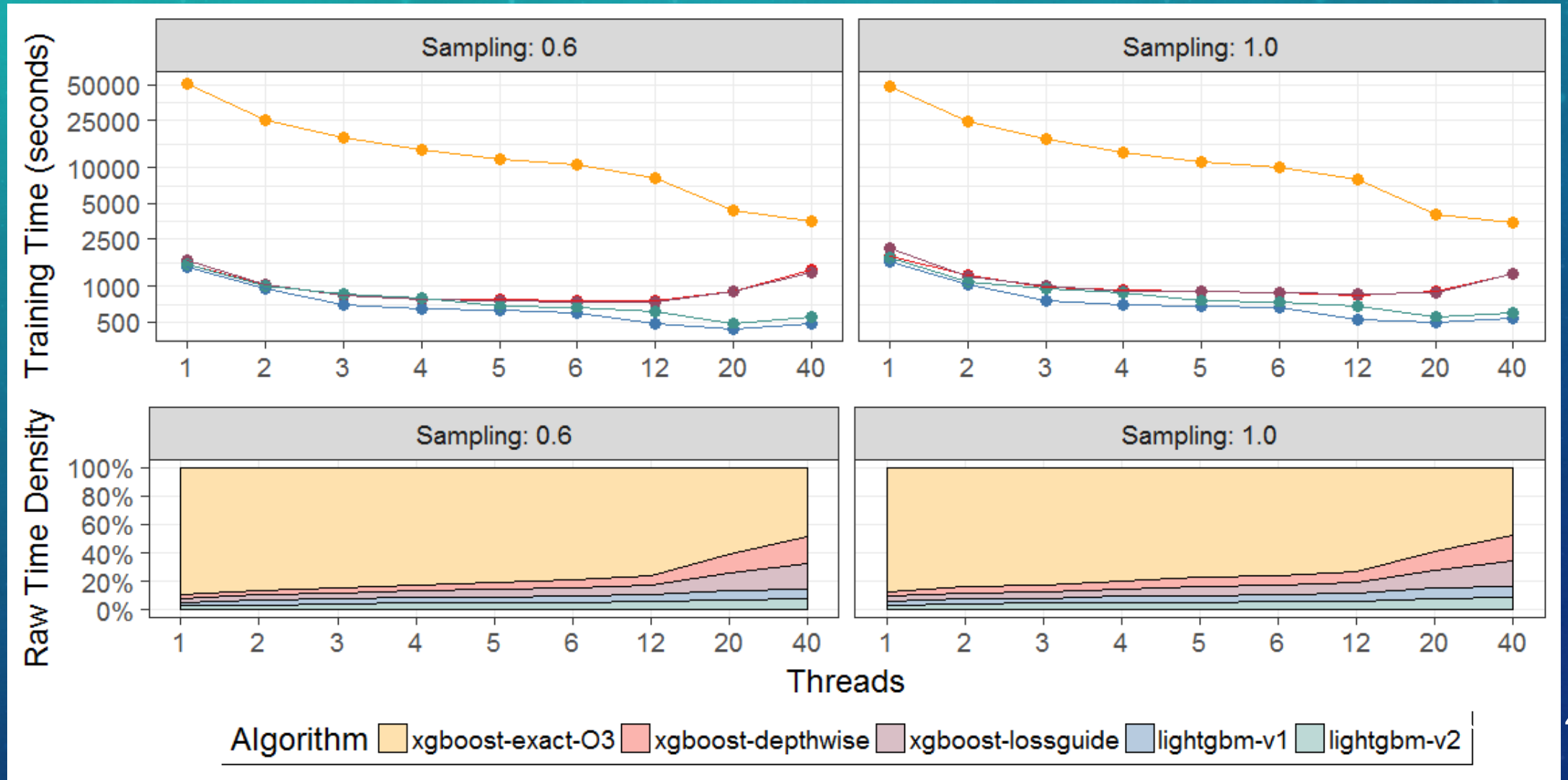
BOSCH SPEED

Time vs Sampling (Stochastic) – Density

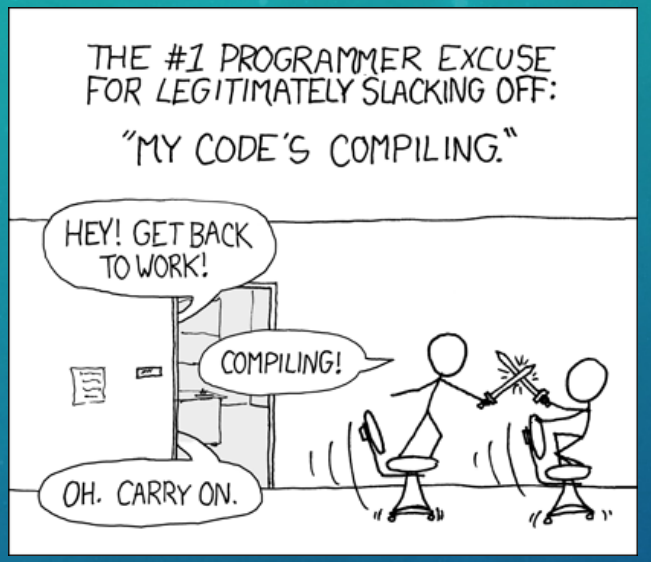


BOSCH SPEED

Time vs Sampling (Stochastic) – Exact



COMPILATION FLAGS TESTED



xgboost

- -O2 -core2
- -O3 -native
- -O3 -fmath –native

Only 3 (was long…)

LightGBM

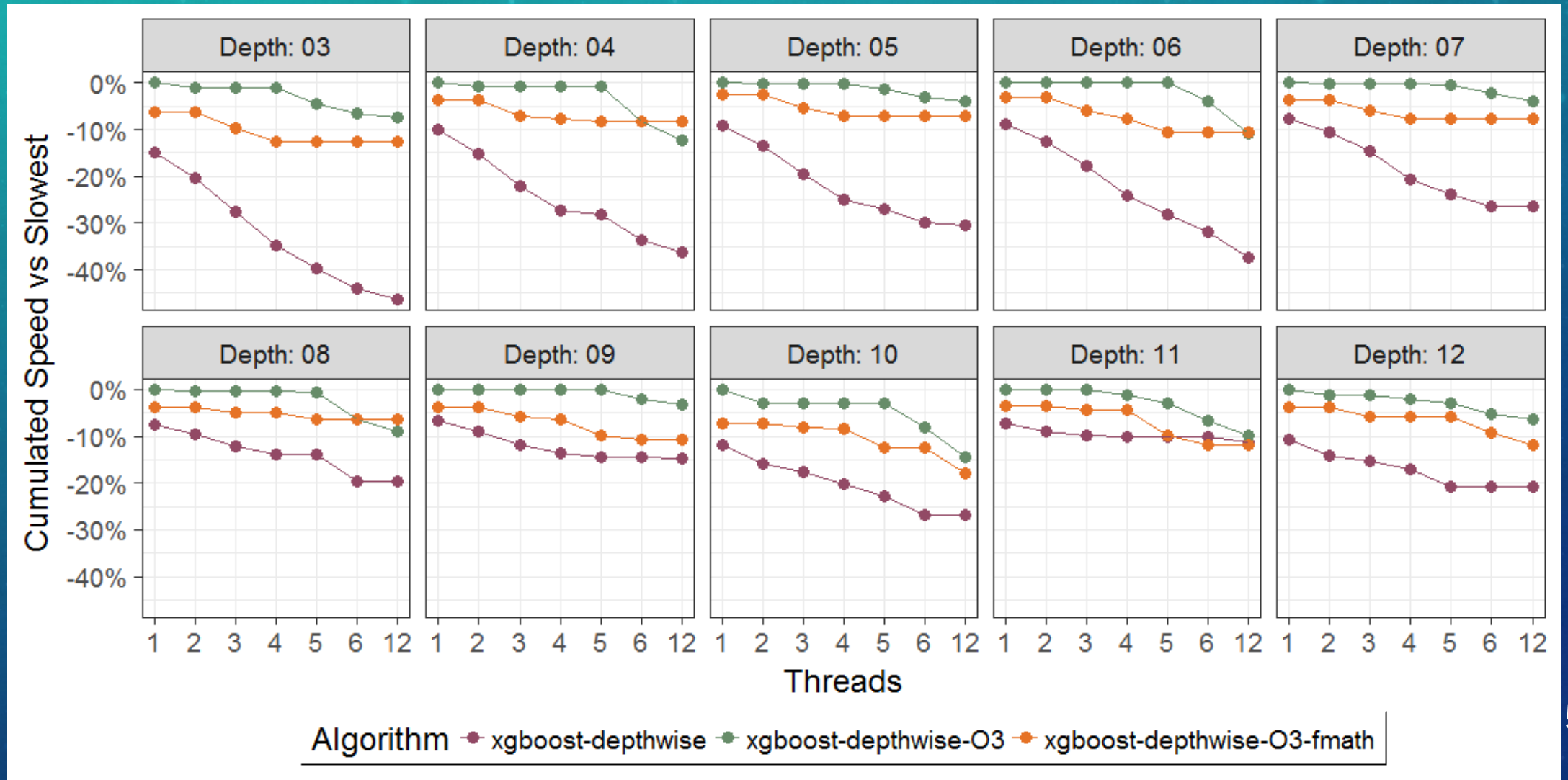
- -Os *–native (-50%)*
- -O2 -core2
- -O2 -native
- -O3 -native
- -O3 -fmath -native

-fmath = -ffast-math

Algolia kaggle

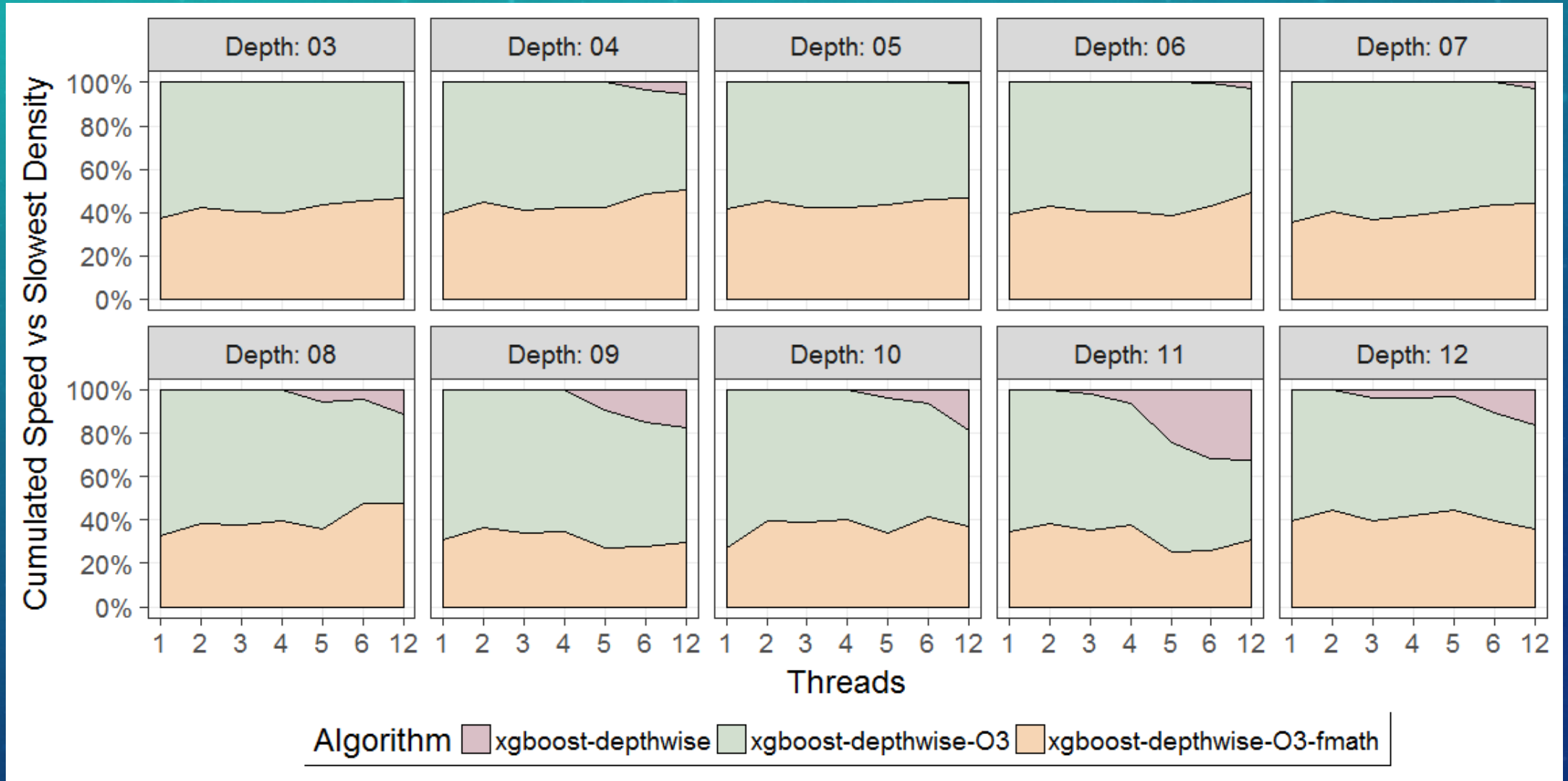
BOSCH SPEED – DEPTH-WISE XGBOOST

Cumulated Time vs Depth – Line



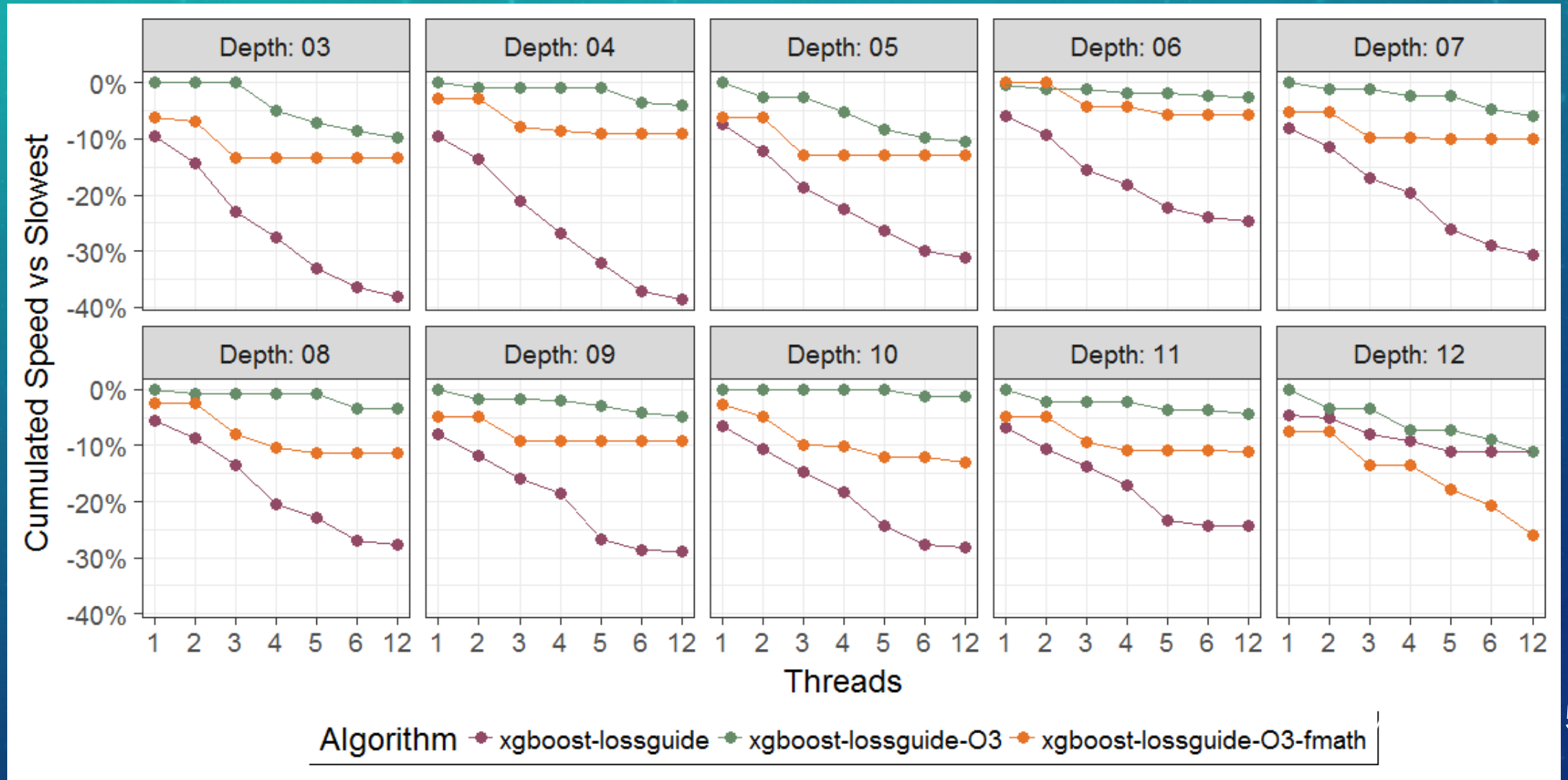
BOSCH SPEED – DEPTH-WISE XGBOOST

Cumulated Time vs Depth – Density



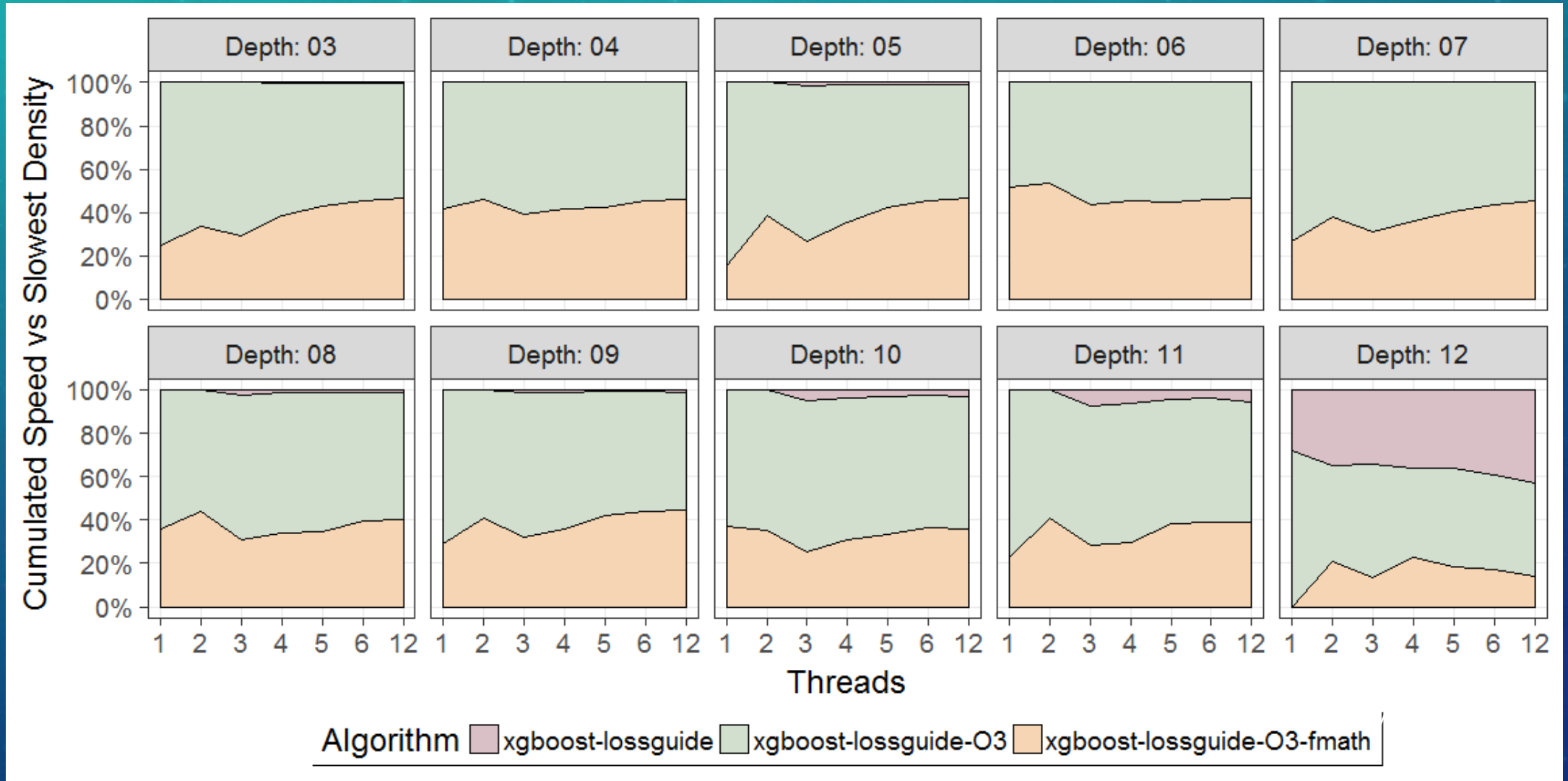
BOSCH SPEED – LOSS GUIDE XGBOOST

Cumulated Time vs Depth – Line



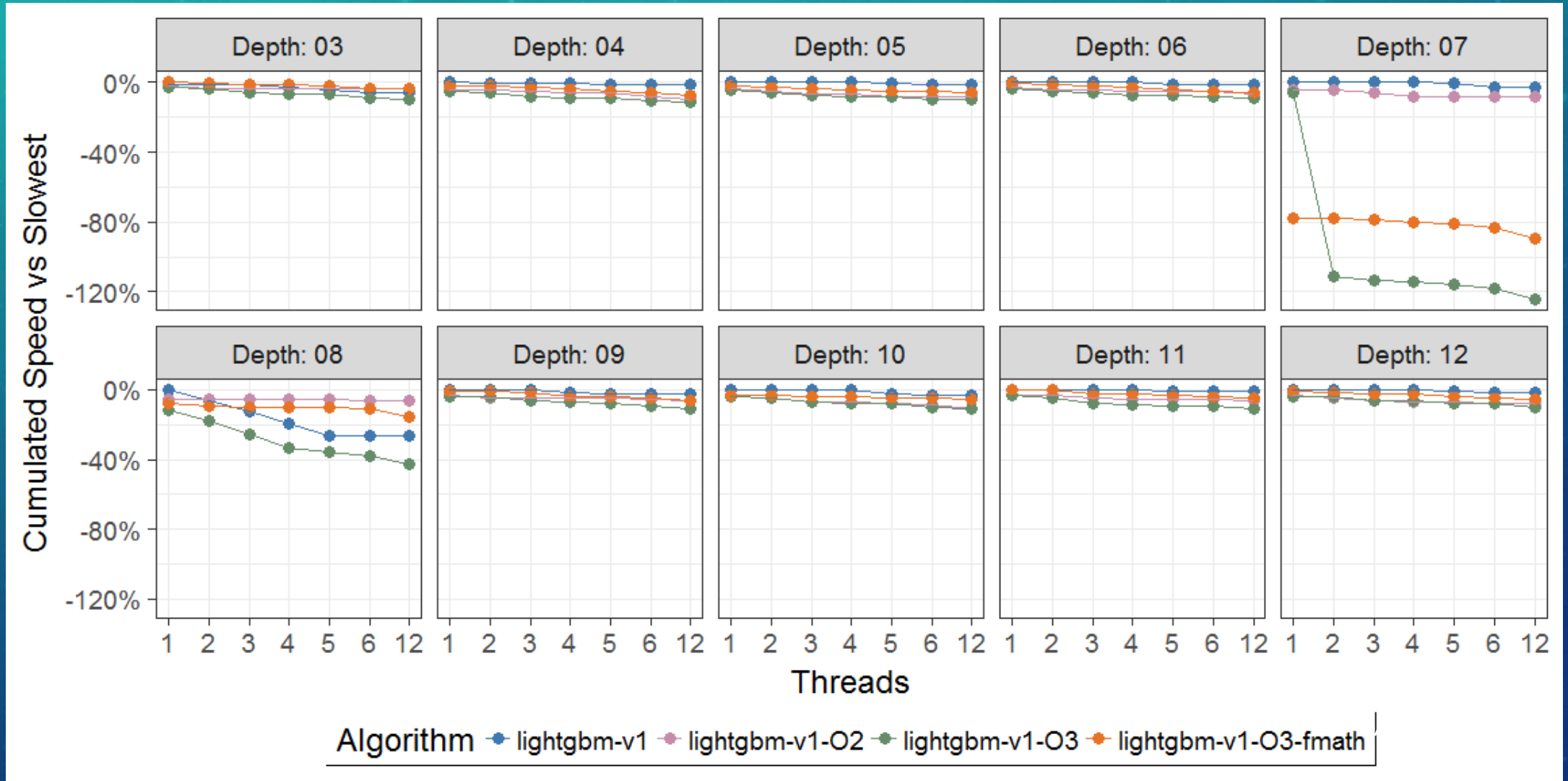
BOSCH SPEED – LOSS GUIDE XGBOOST

Cumulated Time vs Depth – Density



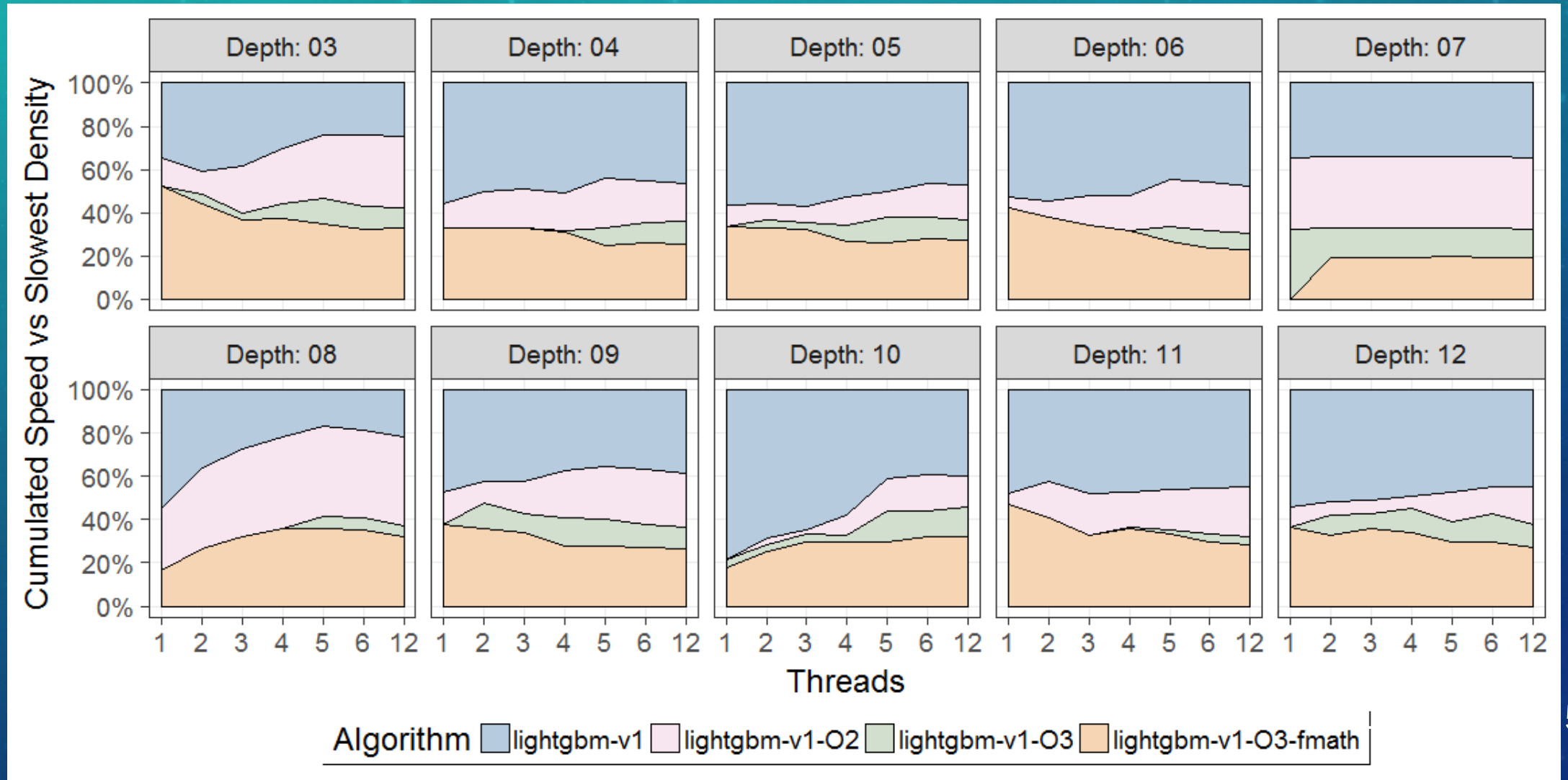
BOSCH SPEED – LIGHTGBM V1

Cumulated Time vs Depth – Line



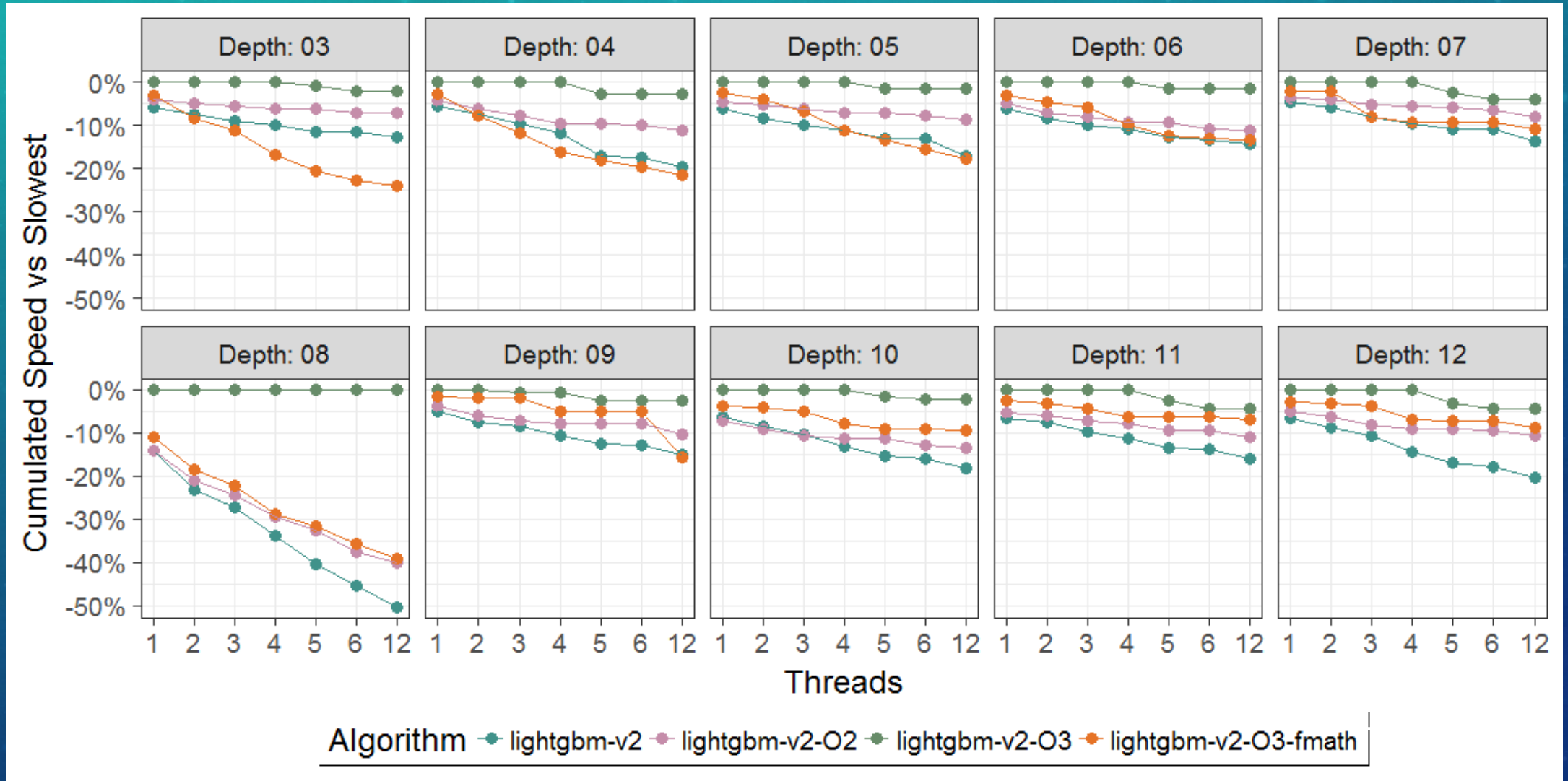
BOSCH SPEED – LIGHTGBM V1

Cumulated Time vs Depth – Density



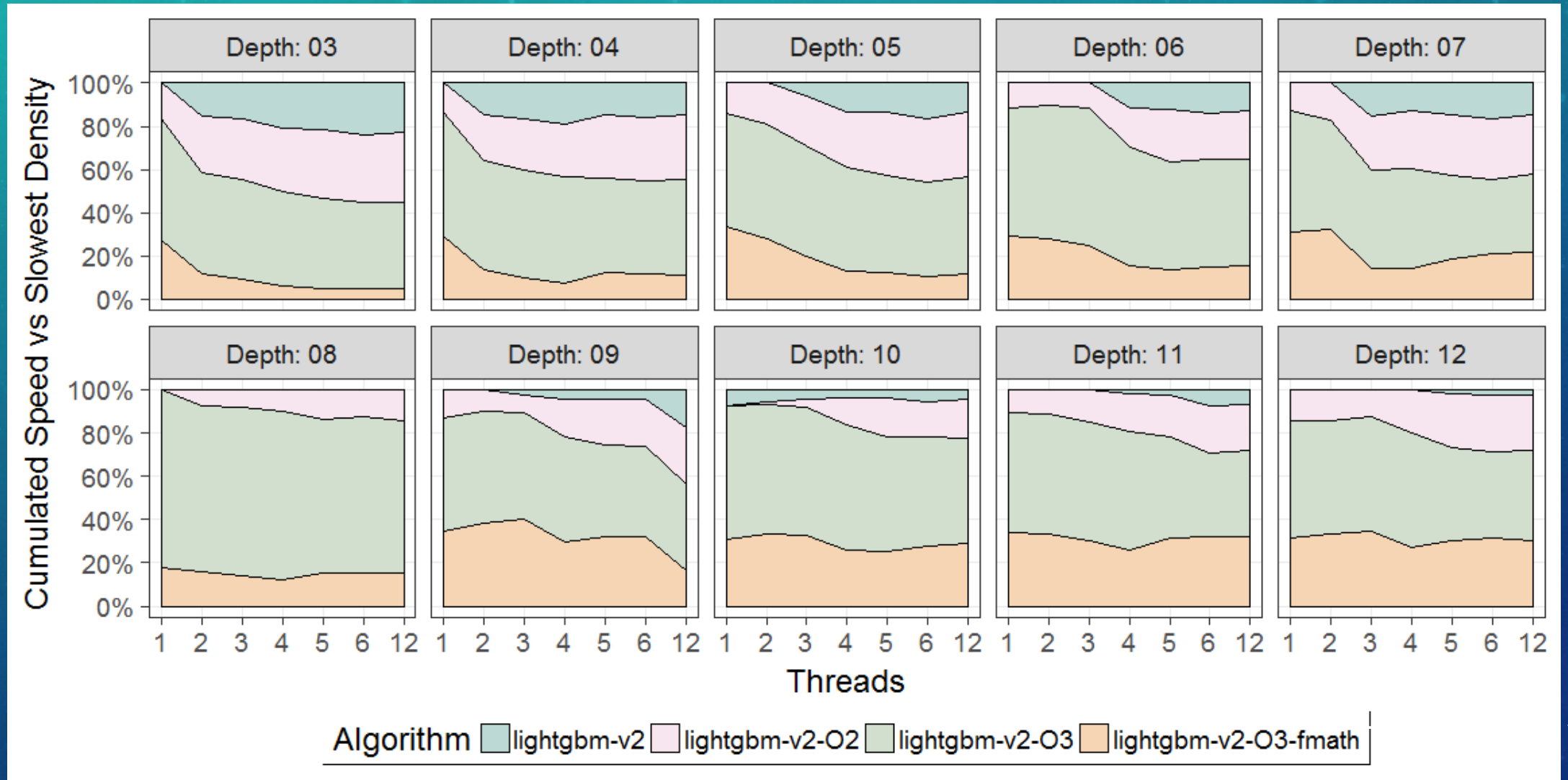
BOSCH SPEED – LIGHTGBM V2

Cumulated Time vs Depth – Line



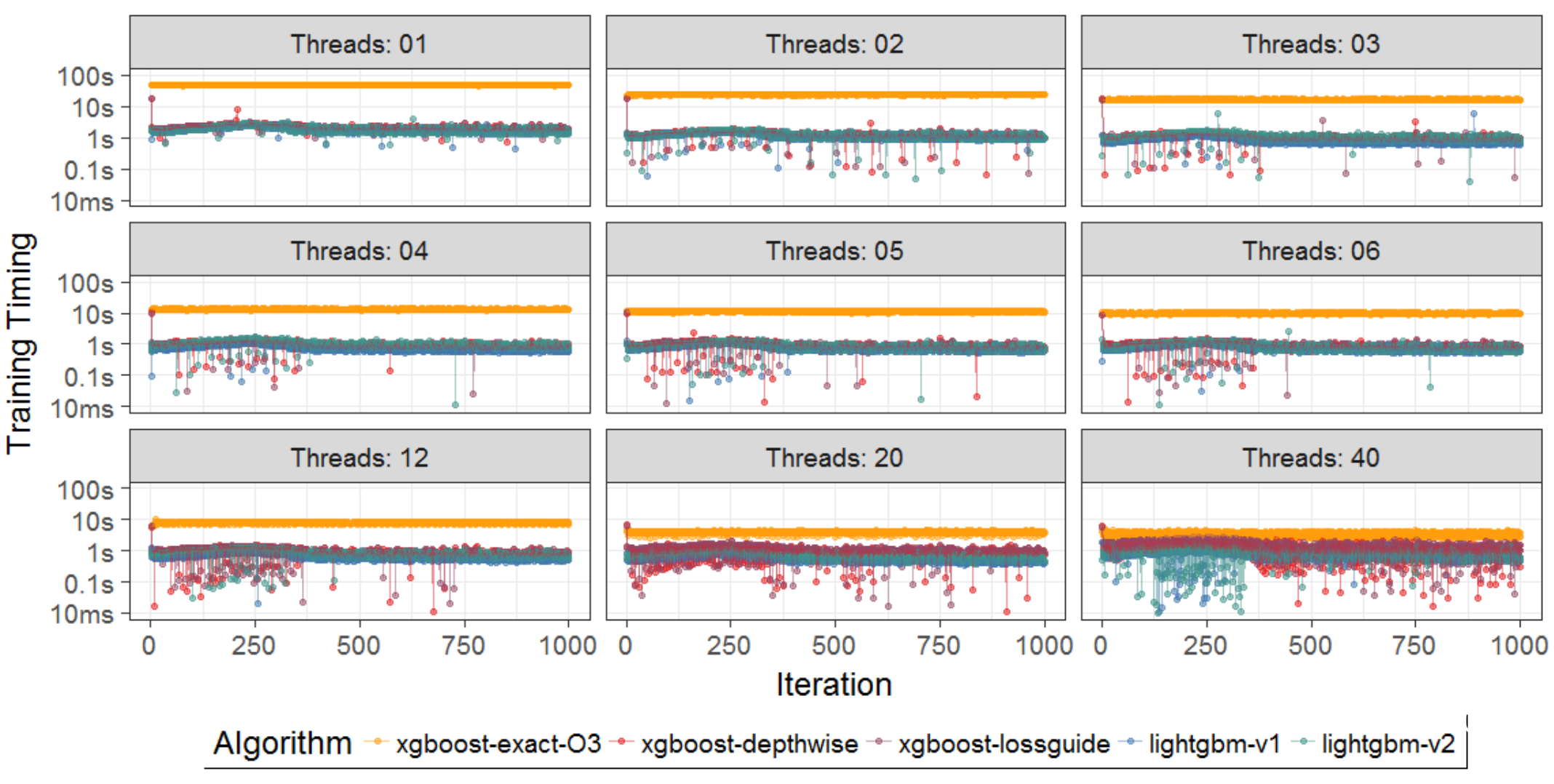
BOSCH SPEED – LIGHTGBM V2

Cumulated Time vs Depth – Density



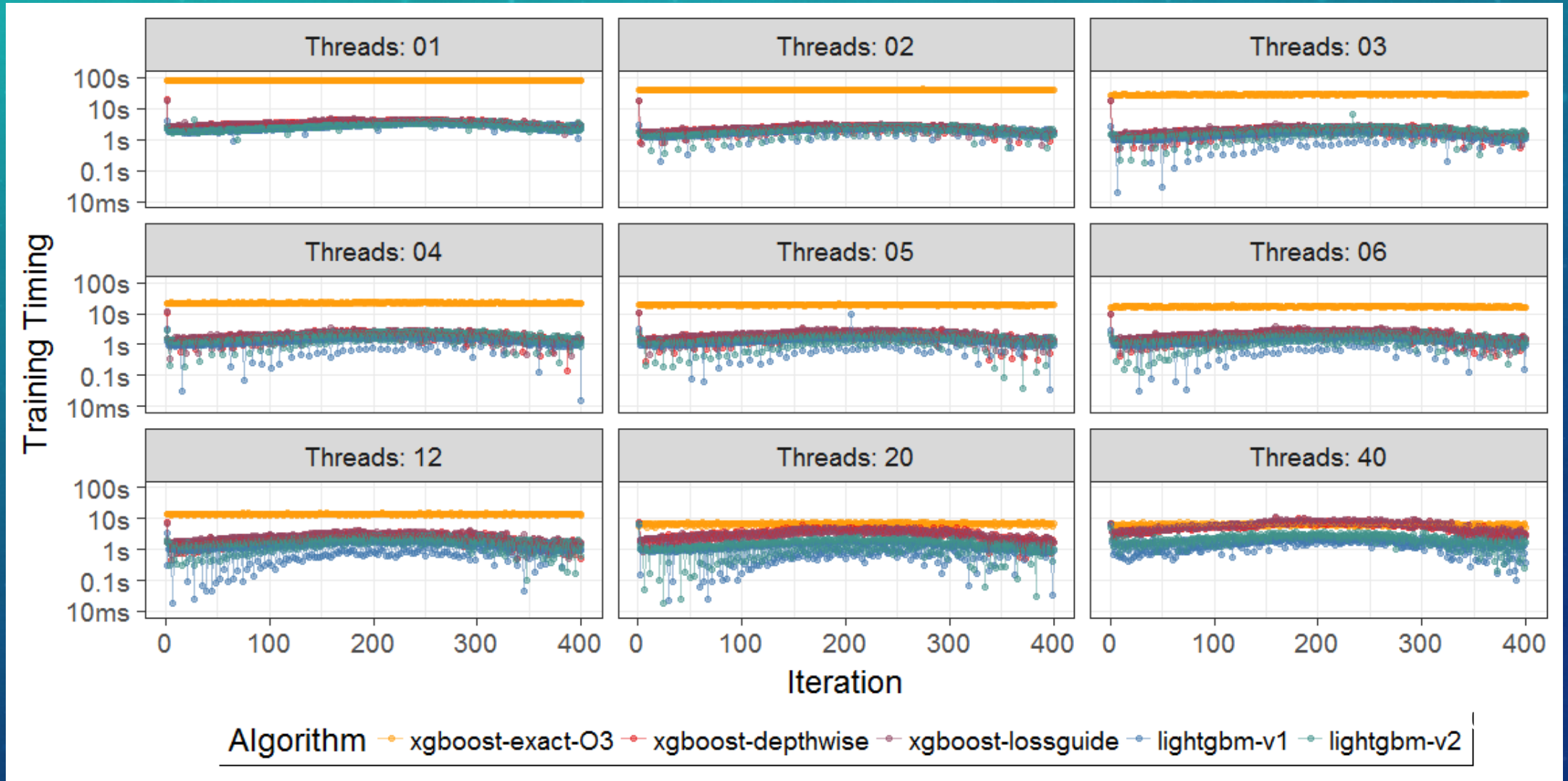
BOSCH SPEED – ZOOM

Zooming on Depth 6



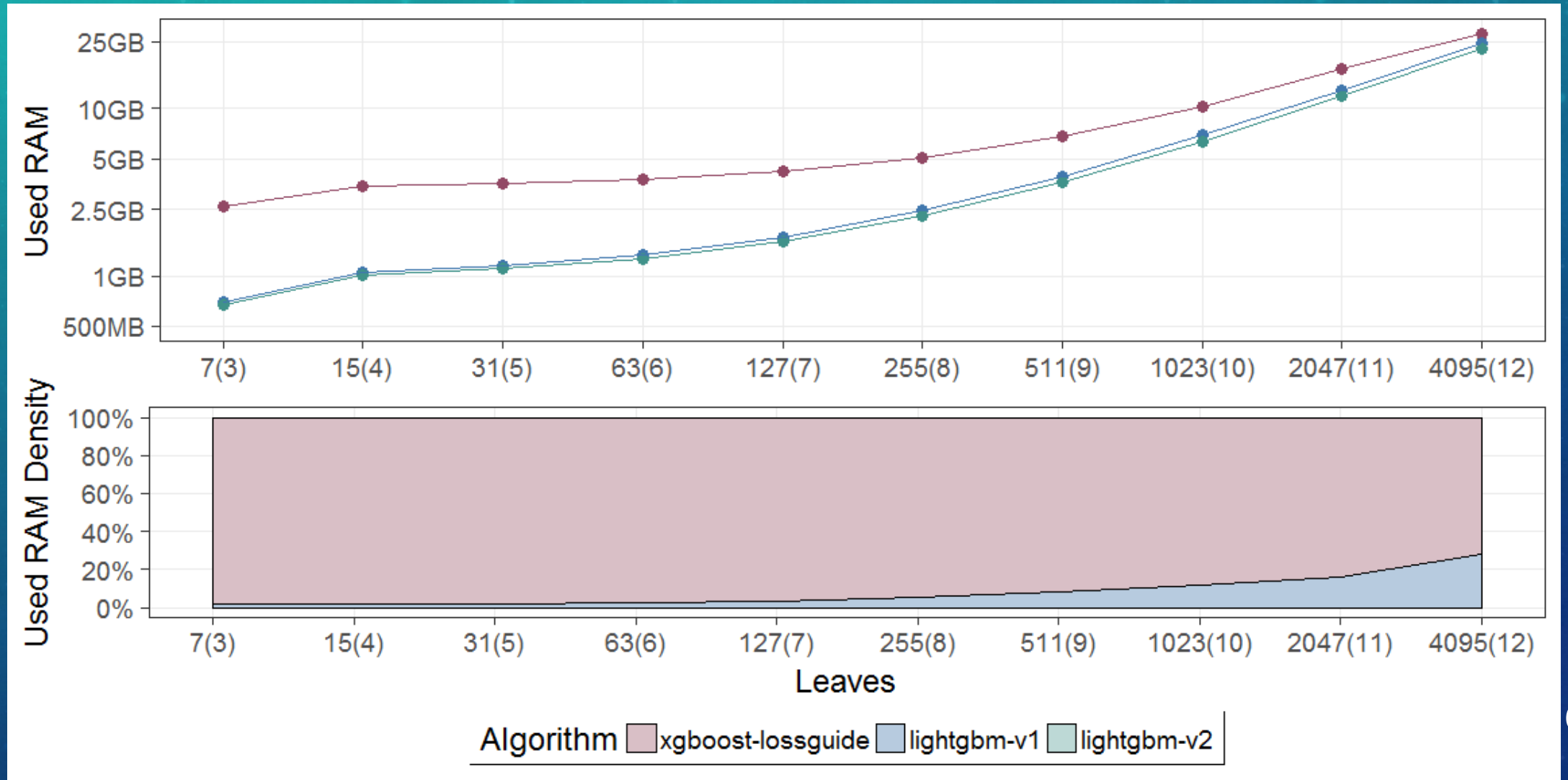
BOSCH SPEED – ZOOM

Zooming on Depth 10



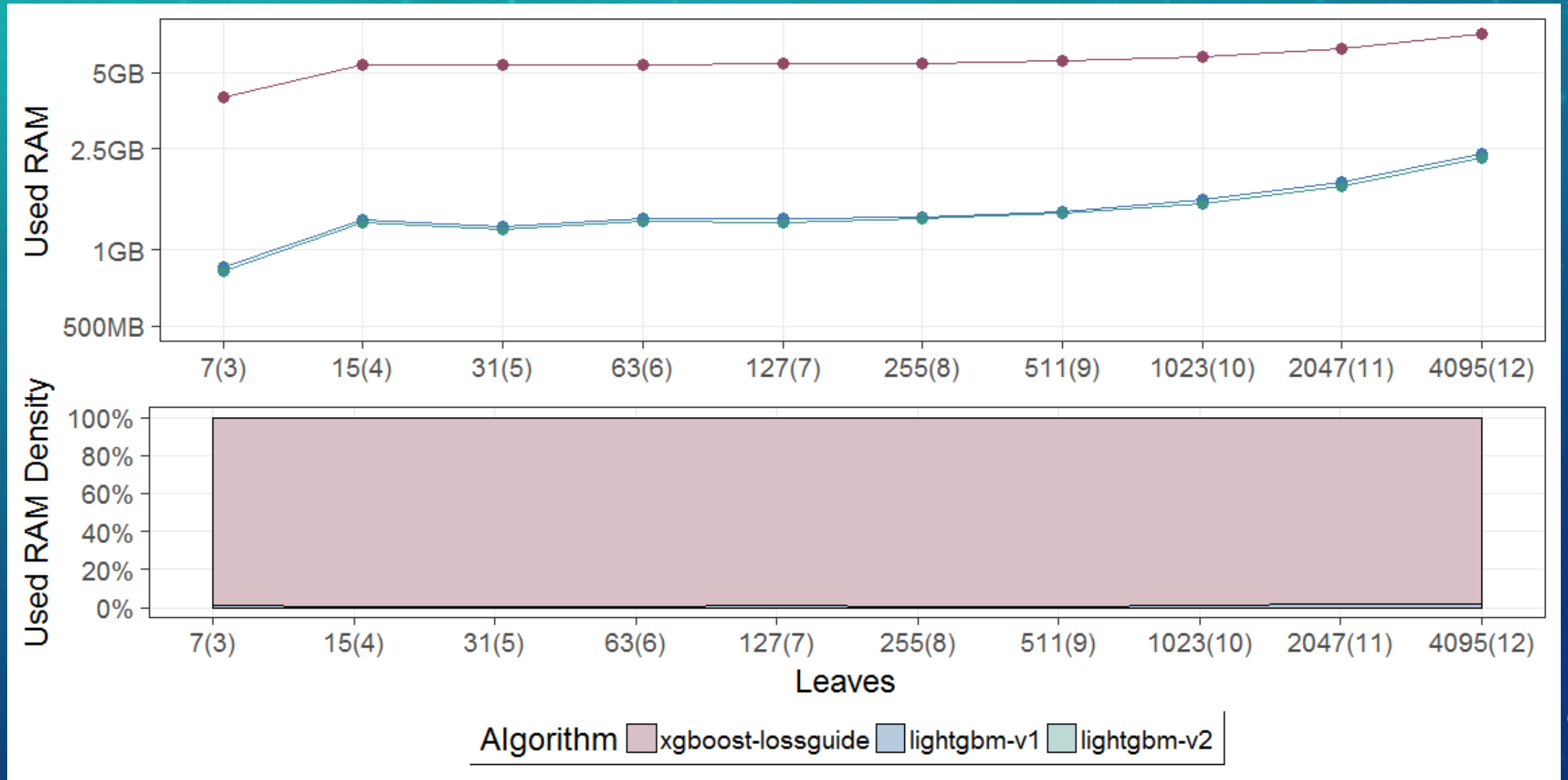
BOSCH EFFICIENCY

Used RAM vs Leaves



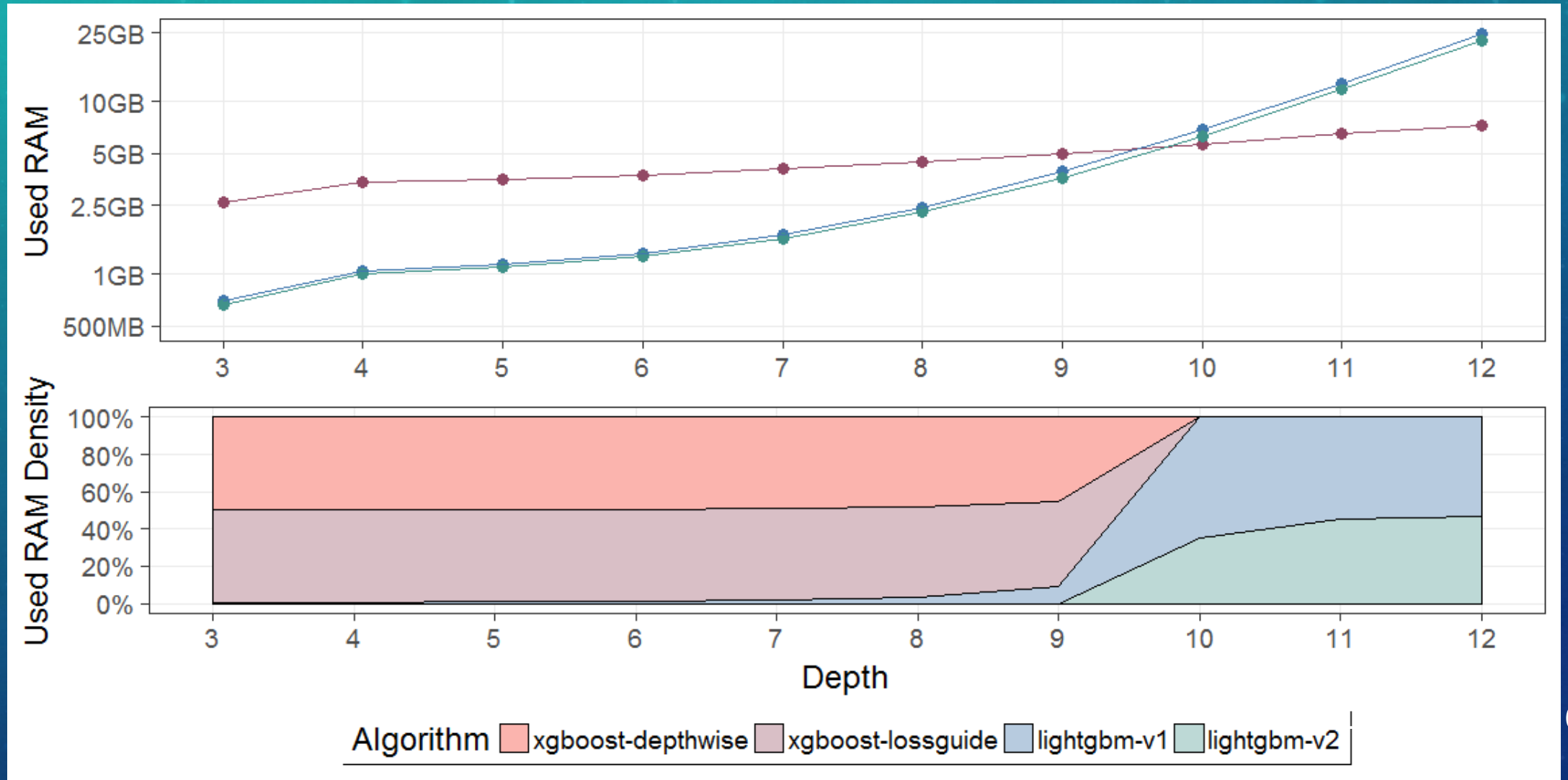
HIGGS EFFICIENCY

Used RAM vs Leaves



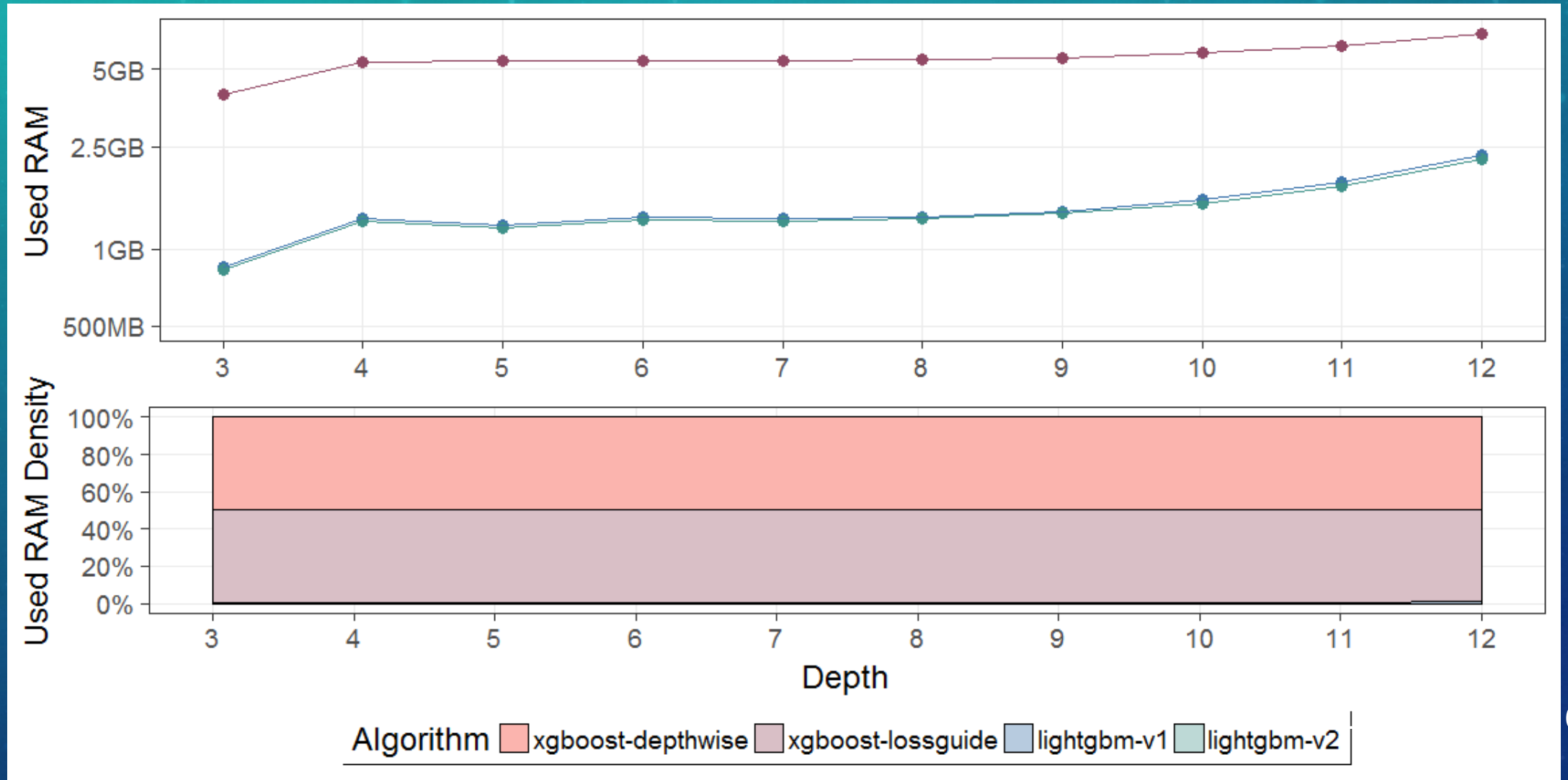
BOSCH EFFICIENCY

Used RAM vs Depth

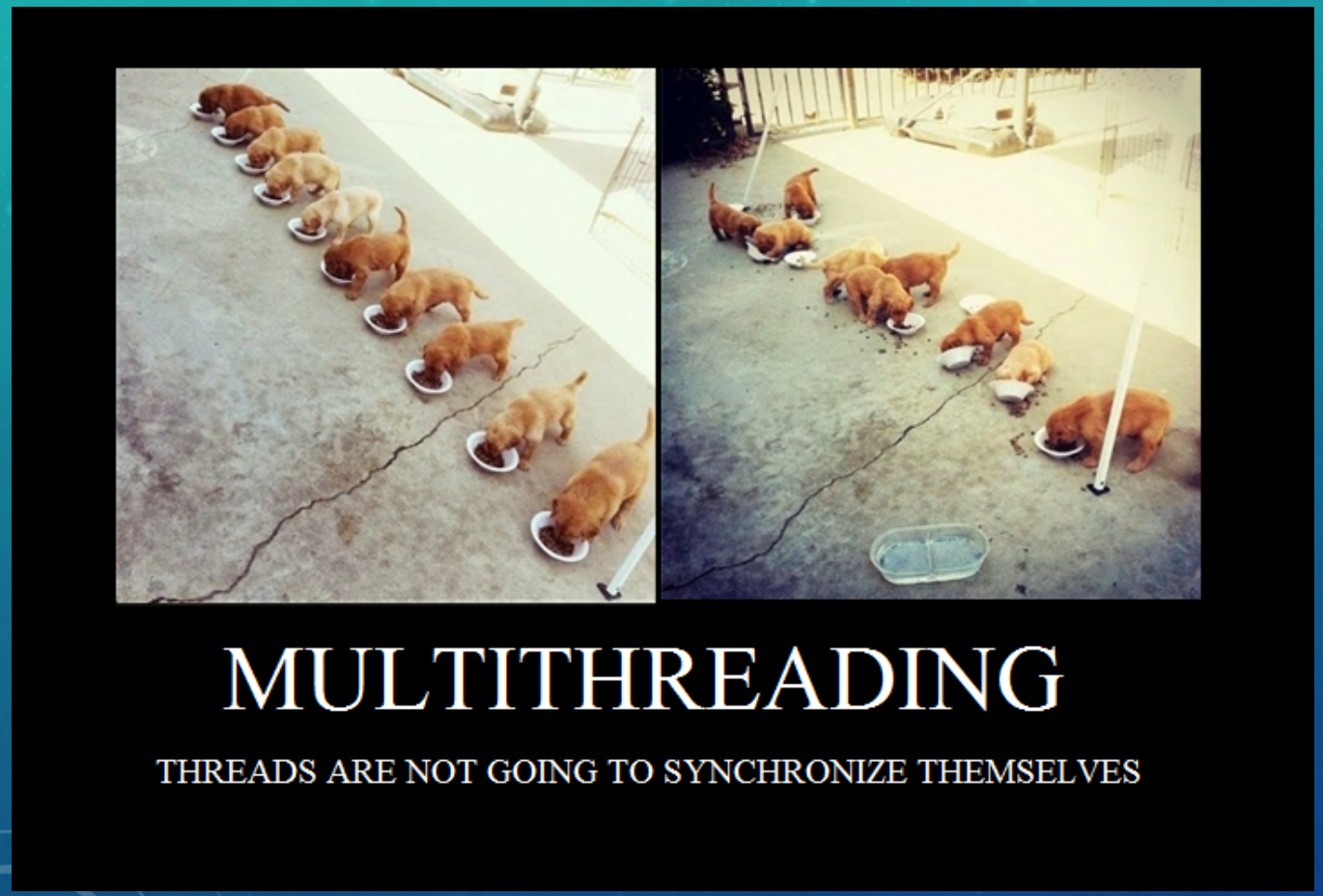


HIGGS EFFICIENCY

Used RAM vs Depth



MULTITHREADING: OVERVIEW



- Time = computation time + overhead
- When overhead is large, less threads must be used:
  - xgboost exact: CPU bound
  - xgboost fast histogram: memory bound
  - LightGBM: memory bound (but less)

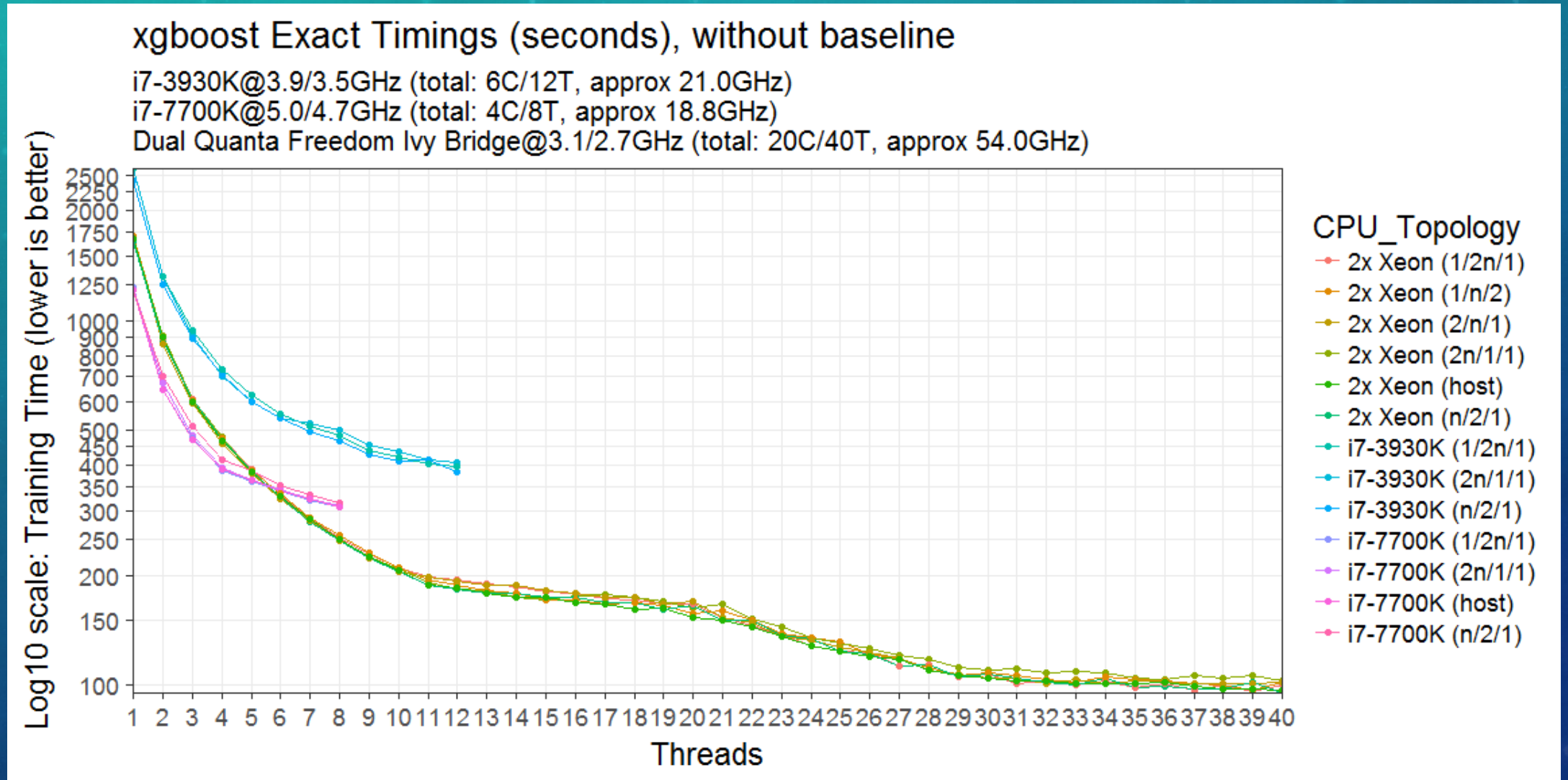
MULTITHREADING: TESTING

- The only way to test multithreading is...

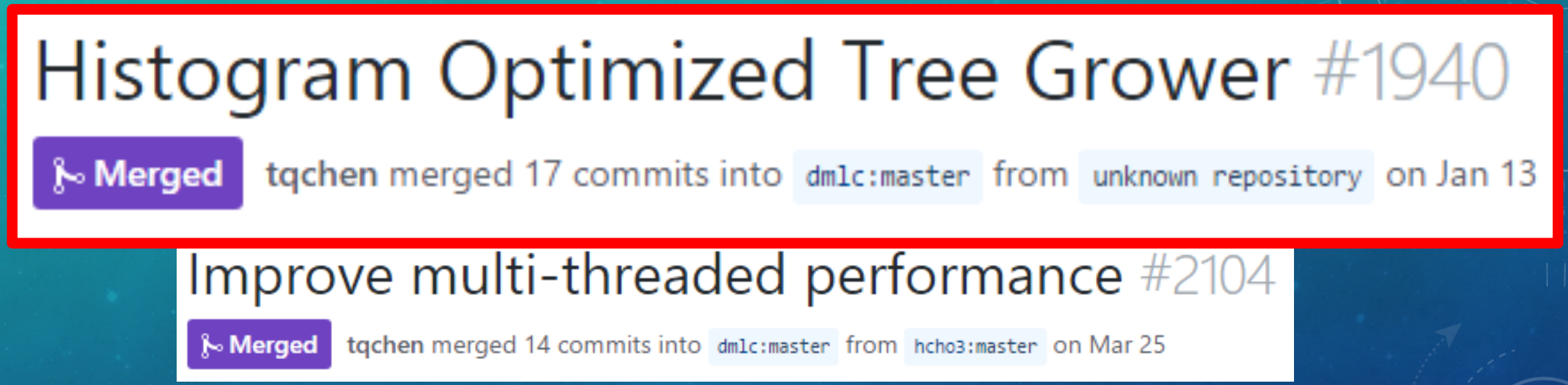
BENCHMARKING

MULTITHREADING

Exact xgboost

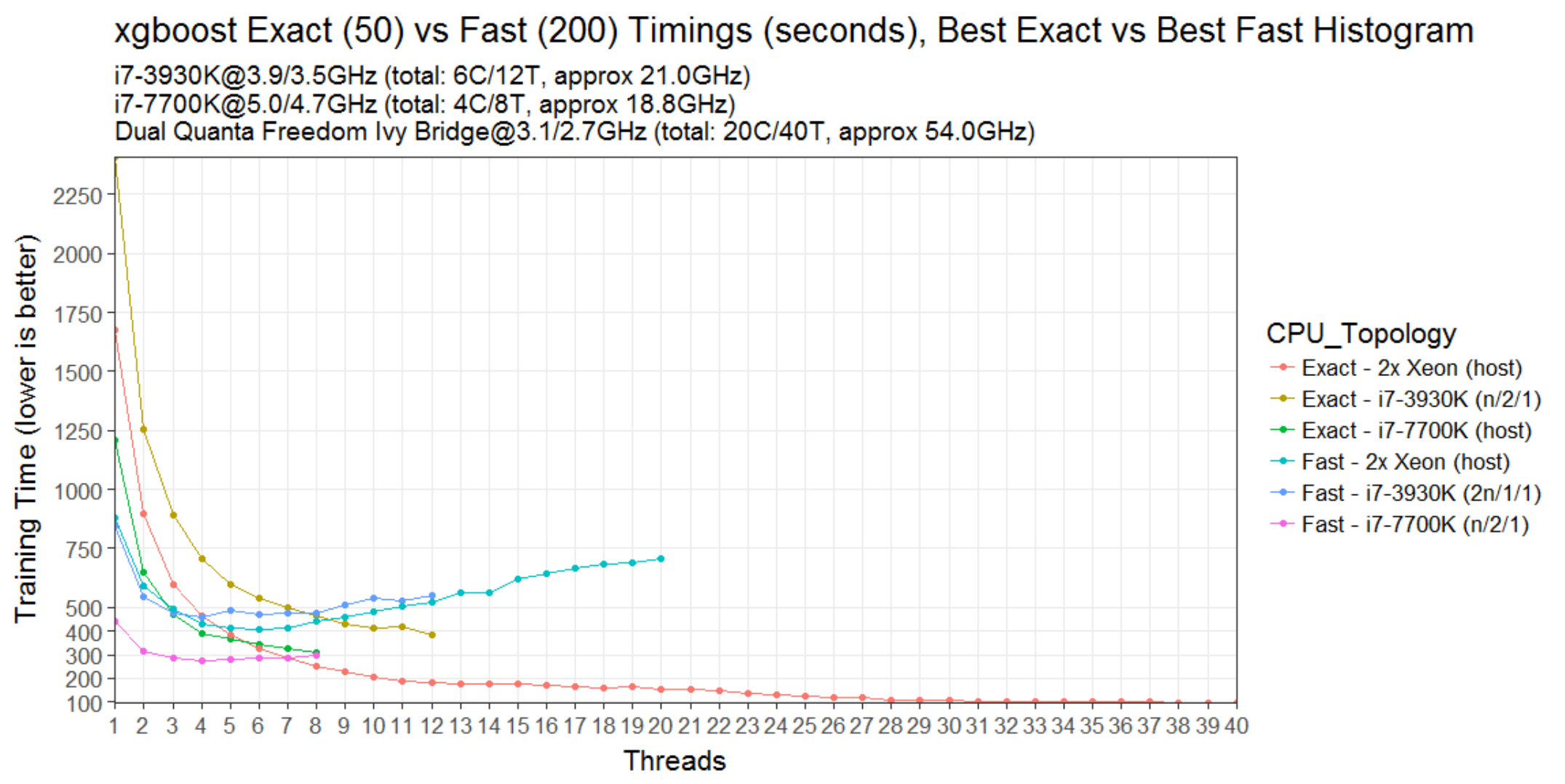


MULTITHREADING: NEW XGBOOST



MULTITHREADING

Fast vs Exact xgboost



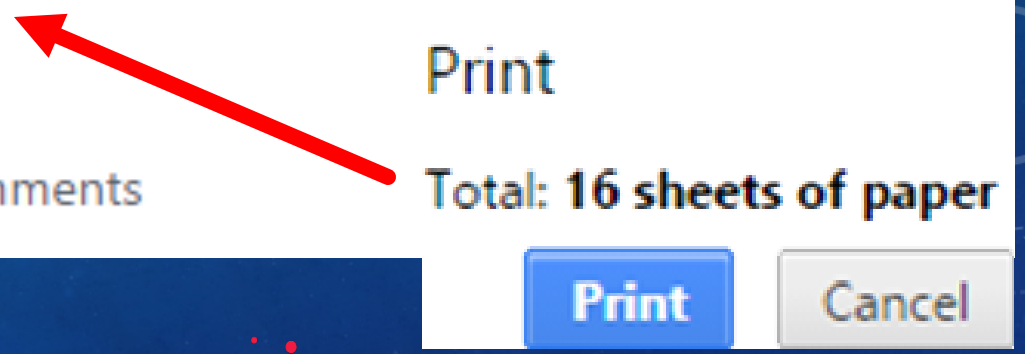
MULTITHREADING: NEW XGBOOST

Improve multi-threaded performance #2104

Merged tqchen merged 14 commits into `dmlc:master` from `hcho3:master` on Mar 25

Since #2104, fast histogram is very slow when multithreaded. #2165

Open Laurae2 opened this issue on Mar 31 · 14 comments



MULTITHREADING: NEW XGBOOST

Since #2104, fast histogram is very slow when multithreaded. #2165

Open Laurae2 opened this issue on Mar 31 · 14 comments

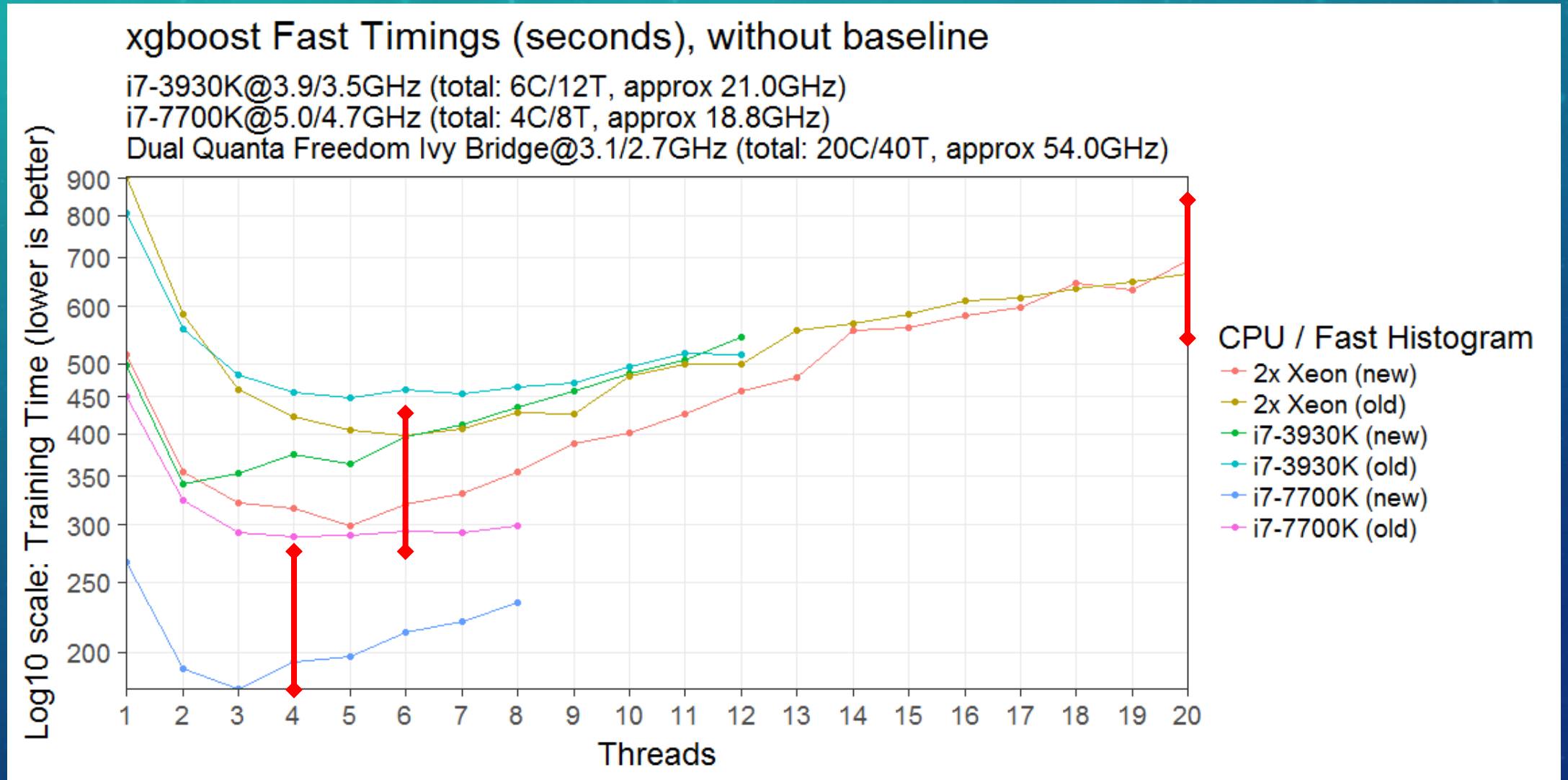
Fix bugs in multithreaded ApplySplitSparseData() #2161

Merged tqchen merged 6 commits into `dmlc:master` from `hcho3:master` on Apr 2

Algolia kaggle

MULTITHREADING

New Fast Histogram xgboost



MULTITHREADING: KNOWLEDGE

- RAM limited for sequential trainings:

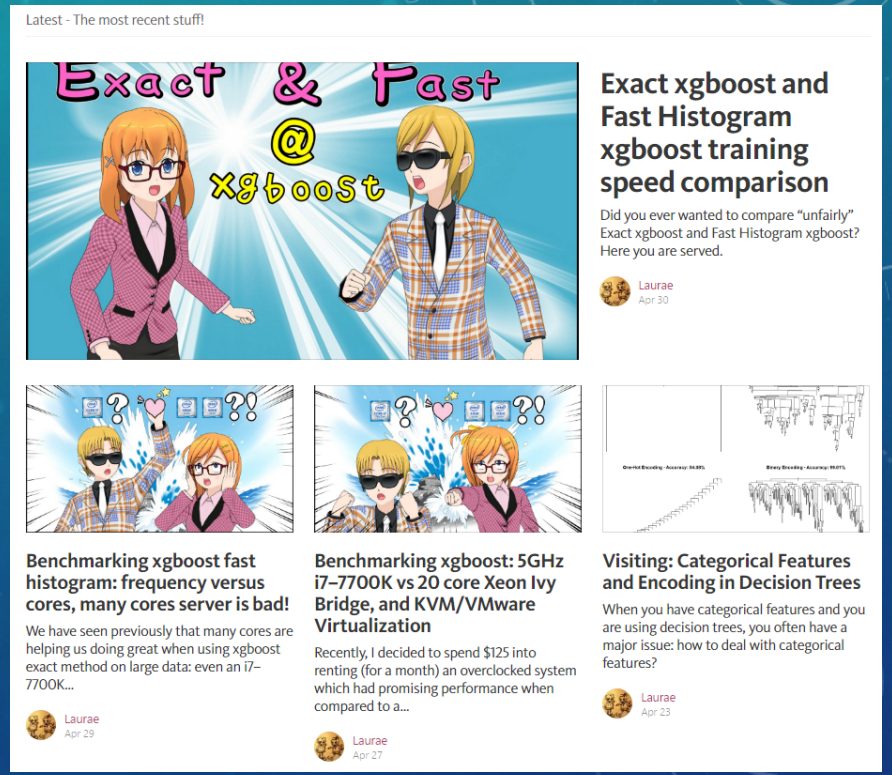
1. Parallelize cross-validation
2. Parallelize training if threads are available
3. Thread count ≤ number of logical cores

MULTITHREADING: KNOWLEDGE

- xgboost:
  - Exact method: small overhead
  - Fast Histogram method: high overhead
  - Dataset dependent (BENCHMARK)

MULTITHREADING: WANT TO SEE MORE?

- https://medium.com/@Laurae2
- https://medium.com/data-design

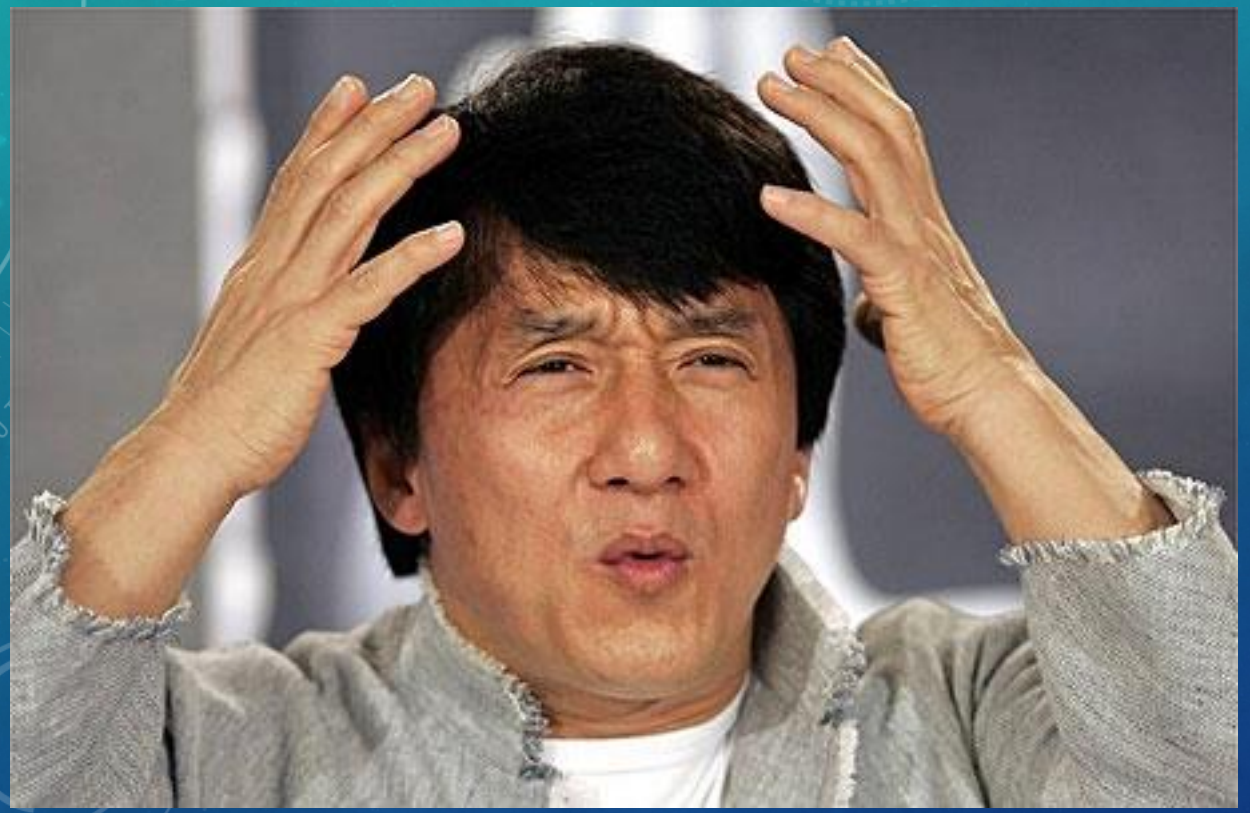


Algolia kaggle

CONCLUSION & FUN FACTS

- xgboost performing great, but *“slow”*
- LightGBM requires appropriate tuning (overfitting faster), but *“fast”*
- Speed is dependent on built trees and on hyperparameters

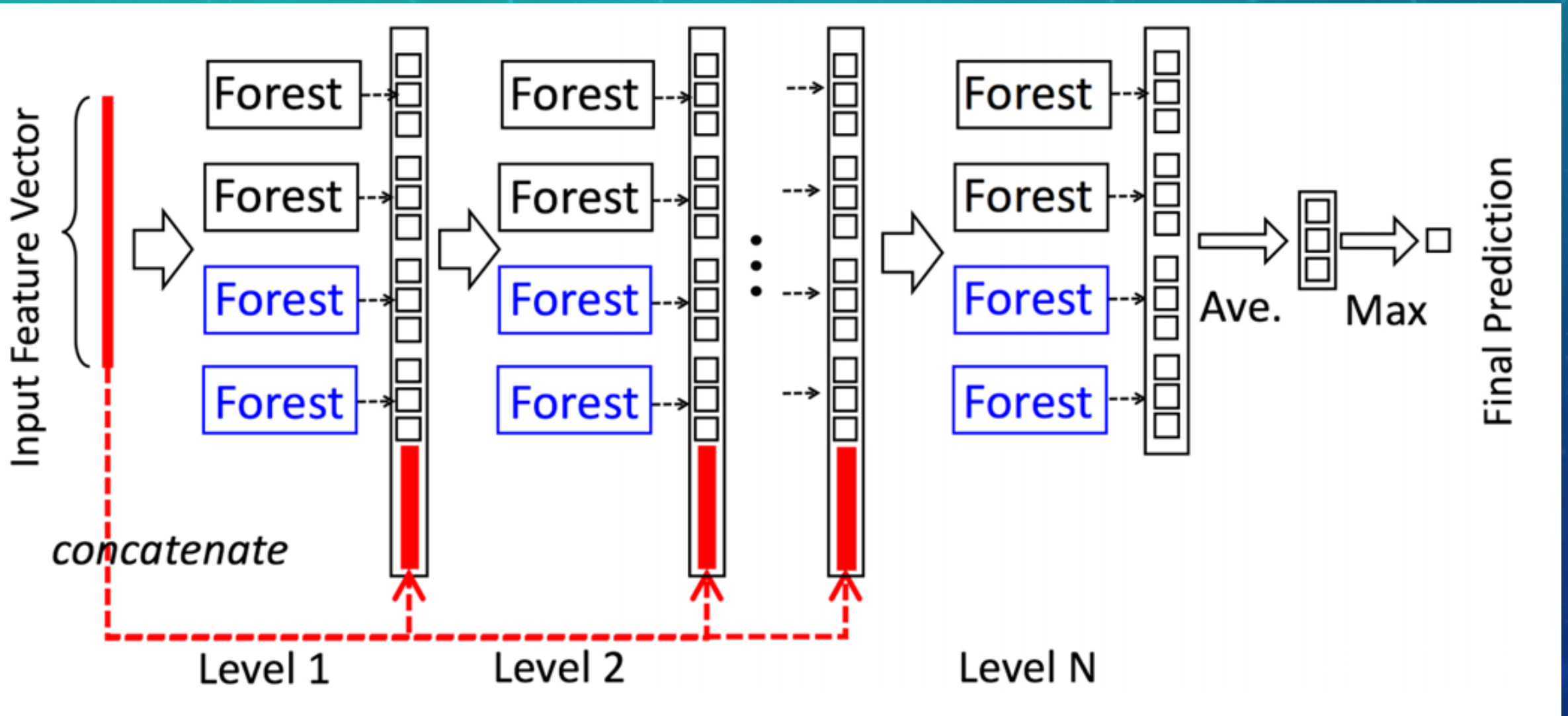
QUESTIONS?



Algolia kaggle

DEEP FOREST

Cascade Forest: Structure



DEEP FOREST

Cascade Forest: Stacking Improvements

Cascade Forest improvements (reported: mean without standard deviation for the forests):

Layer	F1 (RF1)	F2 (RF2)	F3 (CRTF1)	F4 (CRTF2)	Avg Forest	Perf.
Layer 1	86.3972%	86.4238%	78.7237%	77.9620%	85.3142%	=
Layer 2	86.4095%	86.3972%	86.4443%	81.9299%	86.5549%	+
Layer 3	86.5262%	86.4955%	86.6286%	83.9117%	**86.6900%**	Best
Layer 4	86.4976%	86.4750%	84.0305%	83.7930%	86.5856%	-
Layer 5	86.3972%	86.4198%	83.7930%	86.5651%	86.5057%	-
Layer 6	86.3645%	86.3849%	86.4566%	80.6769%	86.4996%	-
Layer 7	86.3993%	86.4300%	80.6769%	86.3993%	86.4627%	-
Layer 8	86.3993%	86.4218%	86.3440%	86.3542%	86.6163%	-

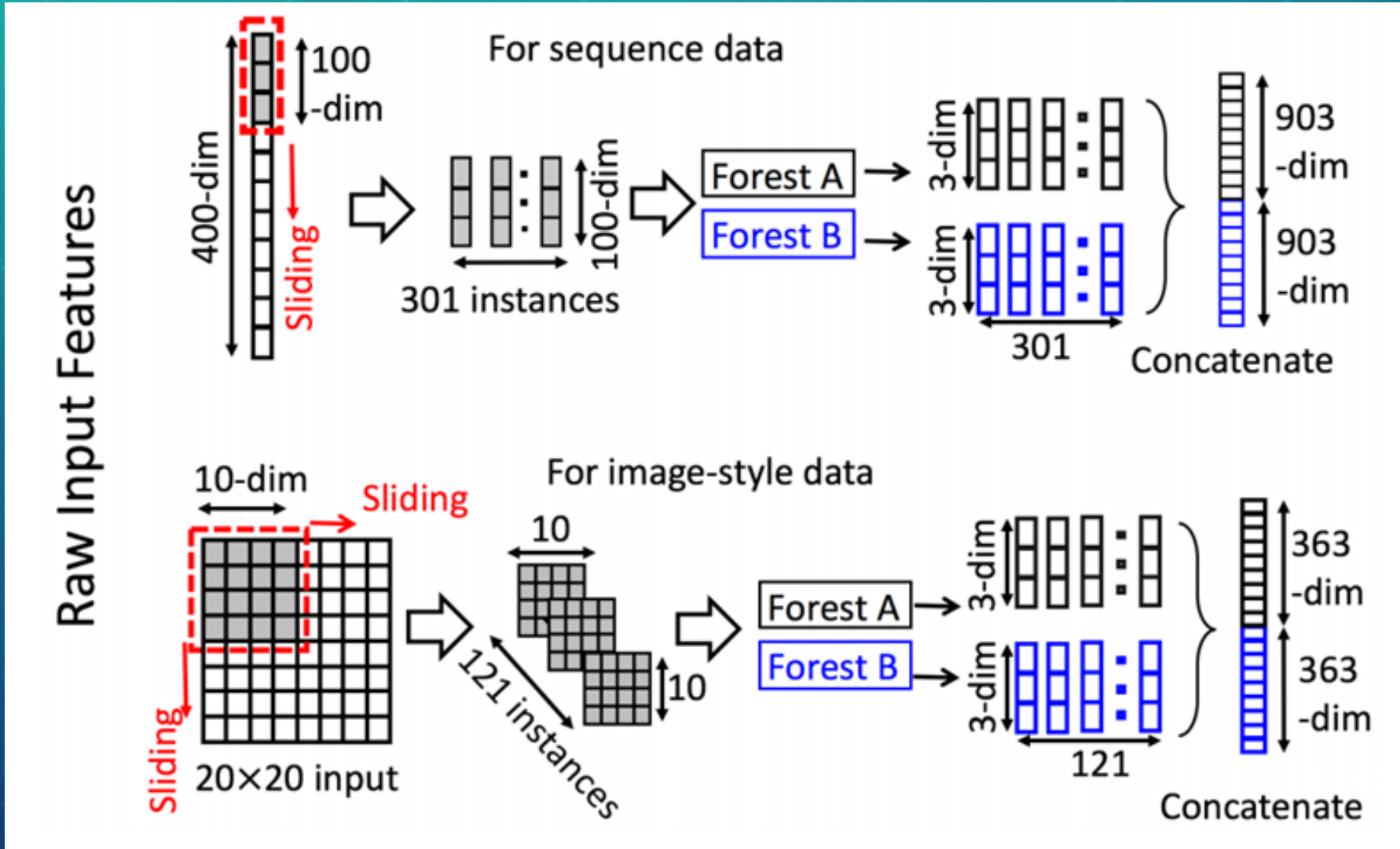
DEEP FOREST

Cascade Forest: Model Sizing Issues

- Model size for ADULT dataset:
  - 8 layers: 2,108,466,480 bytes (1.96GB)
  - 3 layers: 734,047,020 bytes (700MB)

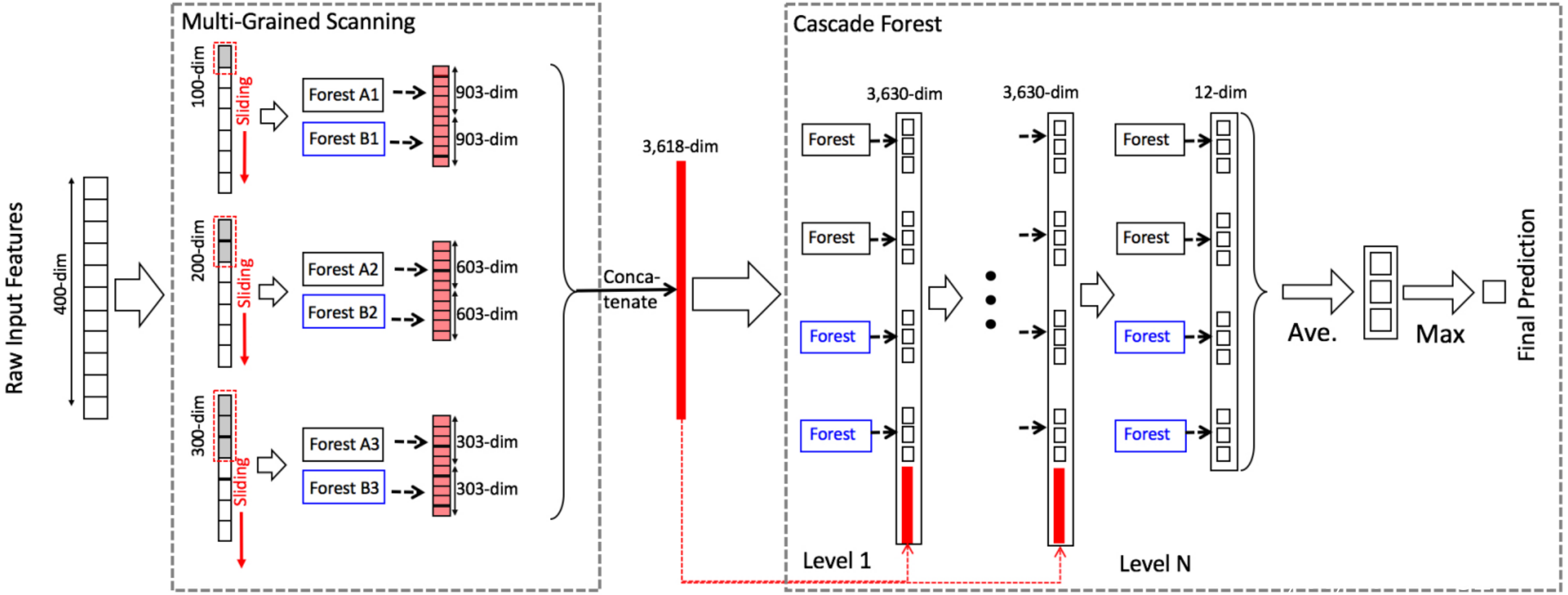
DEEP FOREST

Multi-Grained Scanning Structure



DEEP FOREST

Deep Forest / gcForest Structure



DEEP FOREST

Benchmark: MNIST, 2,000 samples

Benchmark on MNIST 2,000 samples for training, 10,000 samples for testing, i7-4600U, 3-fold cross-validation (Cascade Forest with poor parameters for speed, Multi-Grained Scanning with poor parameters for speed):

Model	Features	Accuracy	Training Time	Model Size
Cascade Forest (xgboost)	784	89.91% 6th iteration	637.264s 11th iteration	Forest: 274,951,008 bytes
Boosted Trees (xgboost)	784	90.53% 250th iteration	267.884s 300 iterations	Boost: NA
"Deep Forest" (xgboost) => Multi-Grained Scanning =>Cascade Forest	Scan: 28x28 Forest: 2404	91.46% 5 iterations	Scan: 449.593s Forest (8): 1135.937s	Scan: 256,419,396 bytes Forest: 273,624,912 bytes
"Deep Boosting" (xgboost) => Multi-Grained Scanning =>Boosted Trees	Scan: 28x28 Boost: 2404	92.41% 215 iterations	Scan: 449.593s Boost (265): 852.360s	Scan: 256,419,396 bytes Boost: NA
LeNet (MXnet + R w/ Intel MKL)	28x28	94.74% 50 epochs	647.638s 50 epochs	CNN: NA

DEEP FOREST

Benchmark: ADULT, all samples

Training time:

Model	Iterations	Accuracy	Time (s)	Perf.
Official paper Deep Forest	see official paper	86.17xx%	?s	6th
xgboost Mode: Random Forest	Full: 2000 iterations	86.5733%	62.001s	5th
xgboost Mode: Boosted Trees	Best: 134 iterations Train: 184 iterations	**87.5868%**	5.737s	1st
Cascade Forest	Best: 3 layers Train: 8 layers	86.6900%	1601.794s	4th
Cascade Forest Stack: Random Forest	Full: 2000 iterations	86.7023%	65.434s	3rd
Cascade Forest Stack: Boosted Trees	Best: 99 iterations Train: 149 iterations	87.2797%	5.983s	2nd

DEEP FOREST

Pros / Cons

- Pros: *“no tuning”* like Random Forest

- Cons:
  - Worse than Gradient Boosting
  - Computation Time / RAM Usage / Model Size

    20GB Matrix + 5.8GB Model for MNIST 70k?

WANT TO KNOW MORE ABOUT PARAMETERS?

https://sites.google.com/view/lauraepp/parameters