Cornell University
Computer Systems Laboratory

RETAIL: OPTING FOR LEARNING SIMPLICITY TO ENABLE QOS-AWARE POWER MANAGEMENT IN THE CLOUD

Shuang Chen,* Angela Jin,** Christina Delimitrou, José F. Martínez

Cornell University
*Currently with Shuhai Lab at Huawei Cloud
** Currently with UCBerkeley

Interactive Latency-Critical (LC) Services

Interactive Latency-Critical (LC) Services

- QoS defined in tail latency (e.g., 99th percentile)

Tail Latency of Interactive LC Applications

- QoS defined in tail latency (e.g., 99th percentile)

Happy == (Latency<=1s)

- QoS defined in tail latency (e.g., 99th percentile)

Happy == (Latency<=1s)

0.1s	Happy
0.1s	Happy
0.1s	Happy
1.9s	Angry
1.9s	Angry
1.9s	Angry

Average: 1s

99th percentile: 1.9s

- **QoS defined in tail latency (e.g., 99^{th} percentile)**

Happy == (Latency<=1s)

Latency	Status
0.1s	Happy
0.1s	Happy
0.1s	Happy
1.9s	Angry
1.9s	Angry
1.9s	Angry

Average: 1s
99^{th} percentile: 1.9s

Latency	Status
1s	Happy
1s	Happy
1s	Happy
1s	Happy
1s	Happy
1s	Happy

Average: 1s
99^{th} percentile: 1s

- **QoS defined in tail latency (e.g., 99^{th} percentile)**

Happy == (Latency<=1s)

Latency	
0.1s	Happy
0.1s	Happy
0.1s	Happy
1.9s	Angry
1.9s	Angry
1.9s	Angry

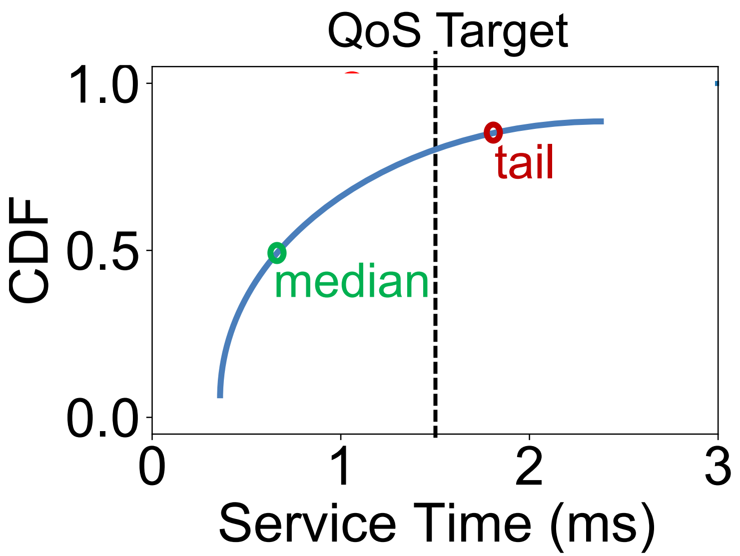
Average: 1s

99^{th} percentile: 1.9s

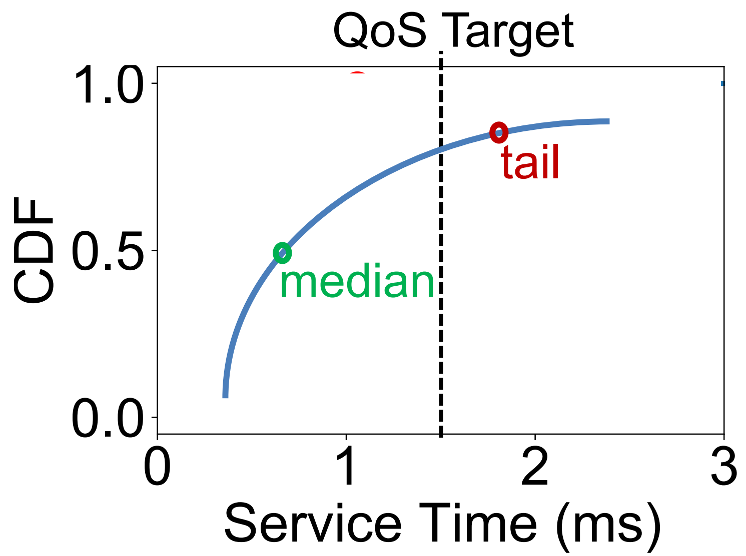
Latency	
1s	Happy
1s	Happy
1s	Happy
1s	Happy
1s	Happy
1s	Happy

Average: 1s

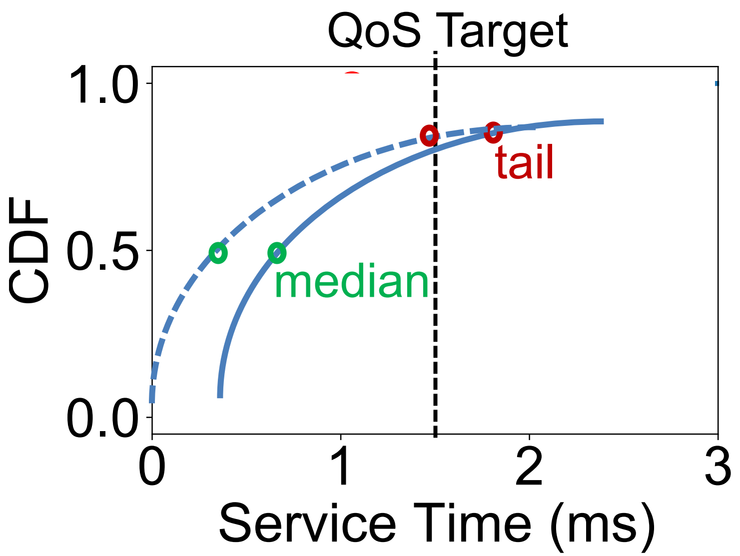
99^{th} percentile: 1s


QoS Target
1.0
0.5
0.0
CDF
tail
median
0
1
2
3
Service Time (ms)

- **Application-level resource management**
  - Conventional resource managers manage each application as a whole

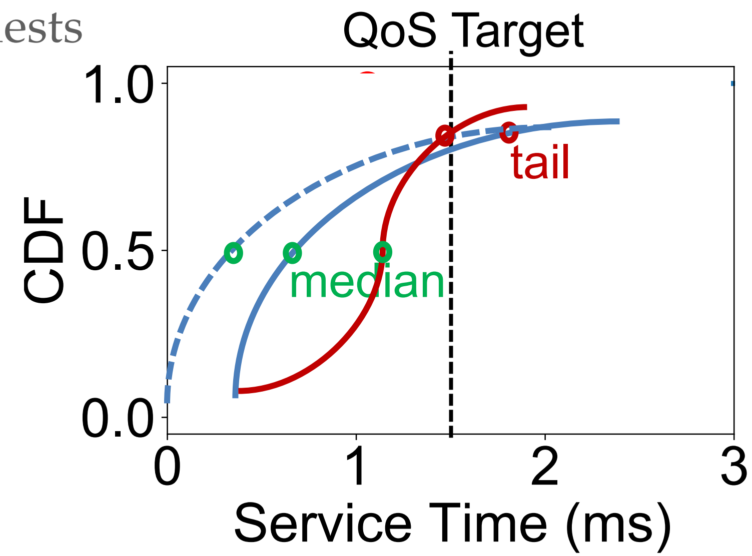


- **Application-level resource management**
  - Conventional resource managers manage each application as a whole

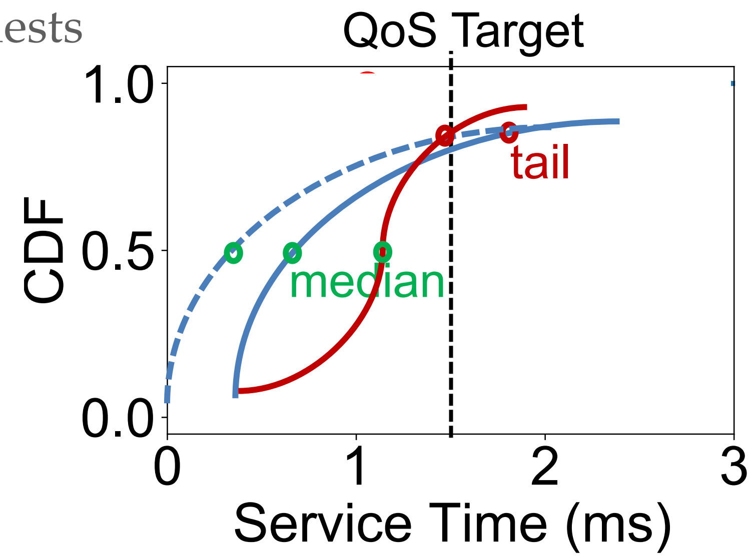


- **Application-level resource management**
  - Conventional resource managers manage each application as a whole

- **Request-level resource management**
  - Make each request *just* meet QoS
    - Assign high frequency to the core running long requests
    - Assign low frequency to the core running short requests
  - Higher resource/power efficiency



- **Application-level resource management**
  - Conventional resource managers manage each application as a whole
- **Request-level resource management**
  - Make each request *just* meet QoS
    - Assign high frequency to the core running long requests
    - Assign low frequency to the core running short requests
  - Higher resource/power efficiency
- **How to know if a request is short or long?**



Prior Work

PRIOR WORK

- **Adrenaline [MICRO’15]**: feature-driven
  - » E.g., if request type is SET, increase frequency
  - ☹ Handpicked features for specific applications
  - ☹ Cannot distinguish requests in the same category

Prior Work

- **Adrenaline [MICRO'15]**: feature-driven
  - » E.g., if request type is SET, increase frequency
  - ☹ Handpicked features for specific applications
  - ☹ Cannot distinguish requests in the same category

- **Gemini [MICRO'20]**: feature-driven, neural-network-based
  - » Predicted latency > QoS, increase frequency
  - ☹ Handpicked features for websearch

PRIOR WORK

- **Adrenaline [MICRO'15]**: feature-driven
  » E.g., if request type is SET, increase frequency
  ☹ Handpicked features for specific applications
  ☹ Cannot distinguish requests in the same category

- **Gemini [MICRO'20]**: feature-driven, neural-network-based
  » Predicted latency > QoS, increase frequency
  ☹ Handpicked features for websearch

Is it possible to predict request latency for a general LC application?

LC APPLICATIONS

Application	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Domain	Key-value store	Image recognition	Speech recognition	Web search	Real-time translation	Database (disk/SSD)	Database (in-memory)
Dataset	One million <key,value> pairs	MNIST [21]	CMU AN4 [11]	English Wikipedia	Spanish articles [6]	TPC-C [16], 1 warehouse	
QoS Target	1ms	5ms	4s	8ms	120ms	5ms	1ms
Median:Tail Ratio	0.84	0.81	0.36	0.27	0.26	0.25	0.19
Request	90% <GET, key> 10% <PUT, key, value>	An image with a handwritten digit	Path to an audio file	A single-word term	A Spanish phrase to be translated into English	47% PAYMENT 45% NEW_ORDER 4% ORDER_STATUS 4% STOCK_LEVEL	
Classification	Little or no variation	Little or no variation	Predicted by request features	Predicted by application features	Predicted by request features	Predicted by request and application features	
Feature(s)	N.A.	N.A.	Audio file size	Document count	Word count	Request type, Item count, Rollback	

- Investigate if it is possible to predict latency for 7 diverse LC applications

LC APPLICATIONS

Application	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Domain	Key-value store	Image recognition	Speech recognition	Web search	Real-time translation	Database (disk/SSD)	Database (in-memory)
Dataset	One million <key,value> pairs	MNIST [21]	CMU AN4 [11]	English Wikipedia	Spanish articles [6]	TPC-C [16], 1 warehouse	
QoS Target	1ms	5ms	4s	8ms	120ms	5ms	1ms
Median:Tail Ratio	0.84	0.81	0.36	0.27	0.26	0.25	0.19
Request	90% <GET, key> 10% <PUT, key, value>	An image with a handwritten digit	Path to an audio file	A single-word term	A Spanish phrase to be translated into English	47% PAYMENT 45% NEW_ORDER 4% ORDER_STATUS 4% STOCK_LEVEL	
Classification	Little or no variation	Little or no variation	Predicted by request features	Predicted by application features	Predicted by request features	Predicted by request and application features	
Feature(s)	N.A.	N.A.	Audio file size	Document count	Word count	Request type, Item count, Rollback	

- Investigate if it is possible to predict latency for 7 diverse LC applications
  - Request latency = service time + queuing delay

LC APPLICATIONS

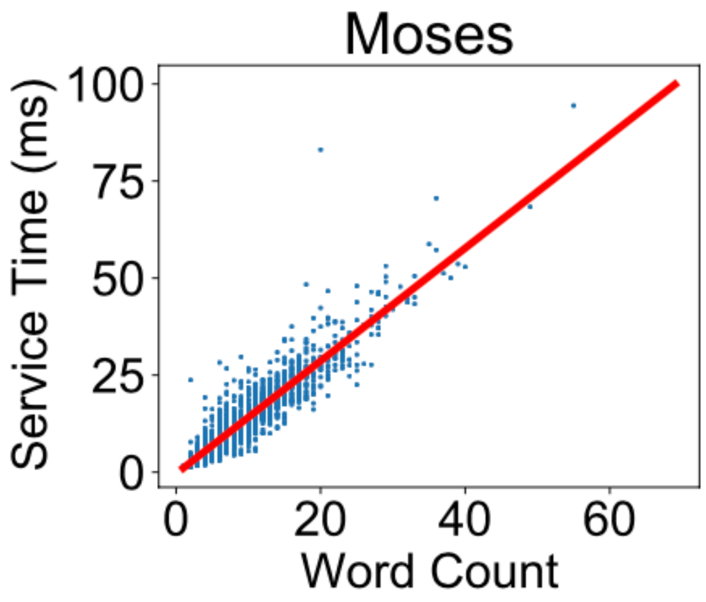
Application	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Domain	Key-value store	Image recognition	Speech recognition	Web search	Real-time translation	Database (disk/SSD)	Database (in-memory)
Dataset	One million <key,value> pairs	MNIST [21]	CMU AN4 [11]	English Wikipedia	Spanish articles [6]	TPC-C [16], 1 warehouse	
QoS Target	1ms	5ms	4s	8ms	120ms	5ms	1ms
Median:Tail Ratio	0.84	0.81	0.36	0.27	0.26	0.25	0.19
Request	90% <GET, key> 10% <PUT, key, value>	An image with a handwritten digit	Path to an audio file	A single-word term	A Spanish phrase to be translated into English	47% PAYMENT 45% NEW_ORDER 4% ORDER_STATUS 4% STOCK_LEVEL	
Classification	Little or no variation	Little or no variation	Predicted by request features	Predicted by application features	Predicted by request features	Predicted by request and application features	
Feature(s)	N.A.	N.A.	Audio file size	Document count	Word count	Request type, Item count, Rollback	

- Investigate if it is possible to predict latency for 7 diverse LC applications
  - Request latency = service time + queuing delay
  - Find features that correlate with service time

- **Request features**
  - Request size, request type, etc.
  - Obtained *at* request arrival

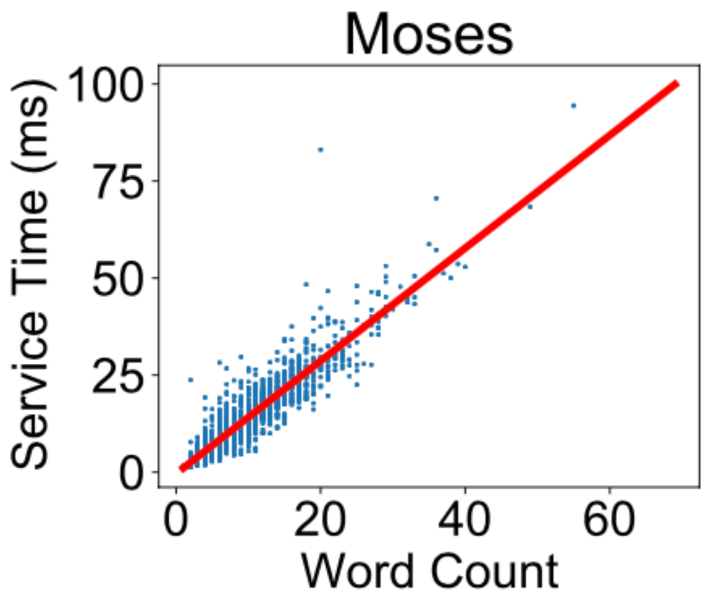
- **Application features**
  - Intermediate variables
  - Obtained *during* request processing

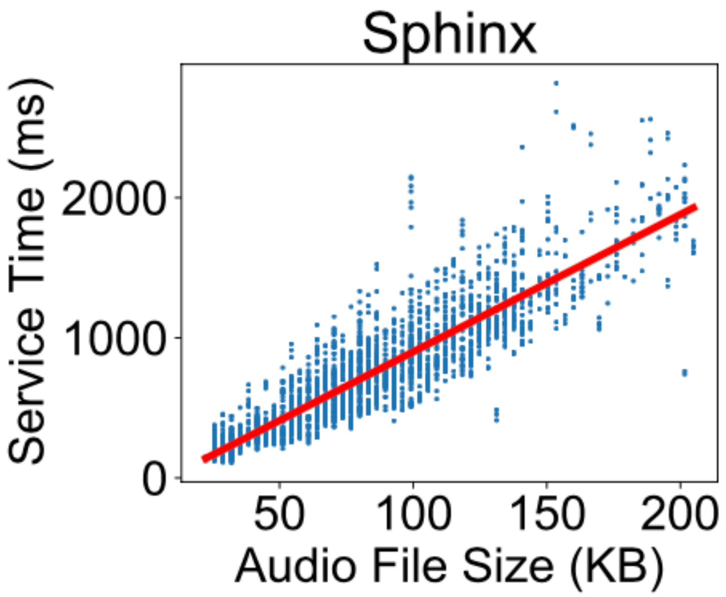
Characterization – Request Size



- Real-time translation
- Input request:
  a Spanish phrase

Characterization – Request Size

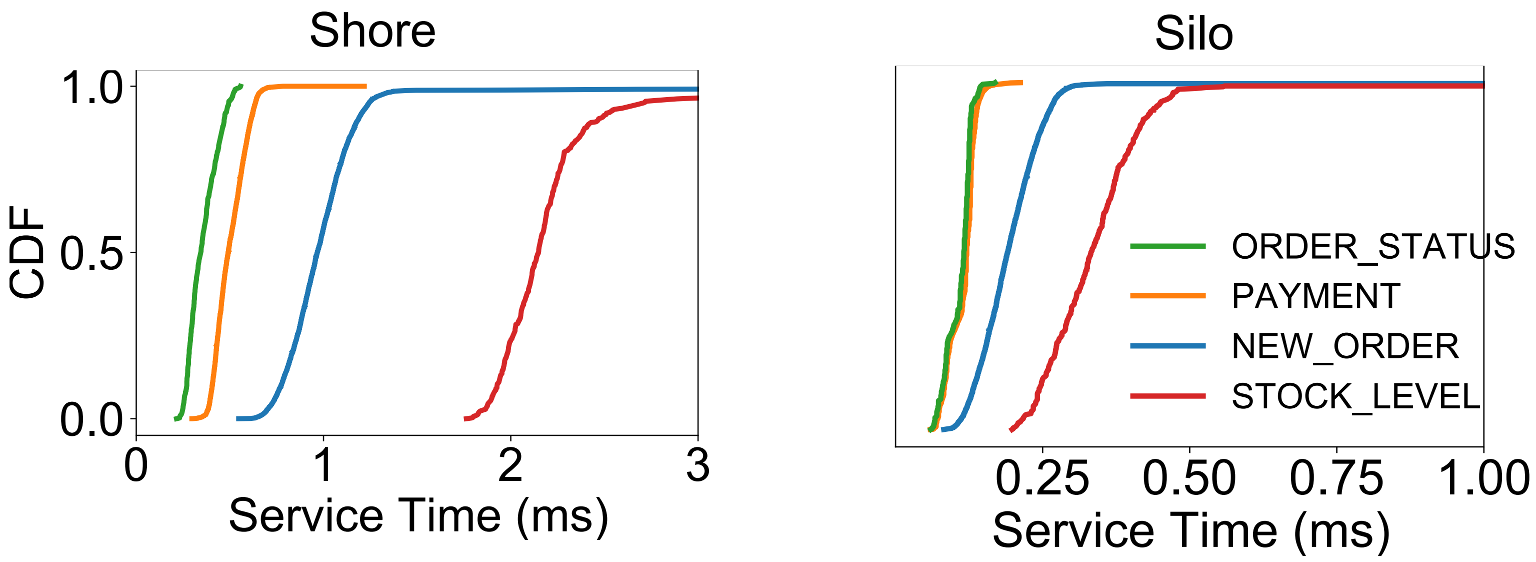




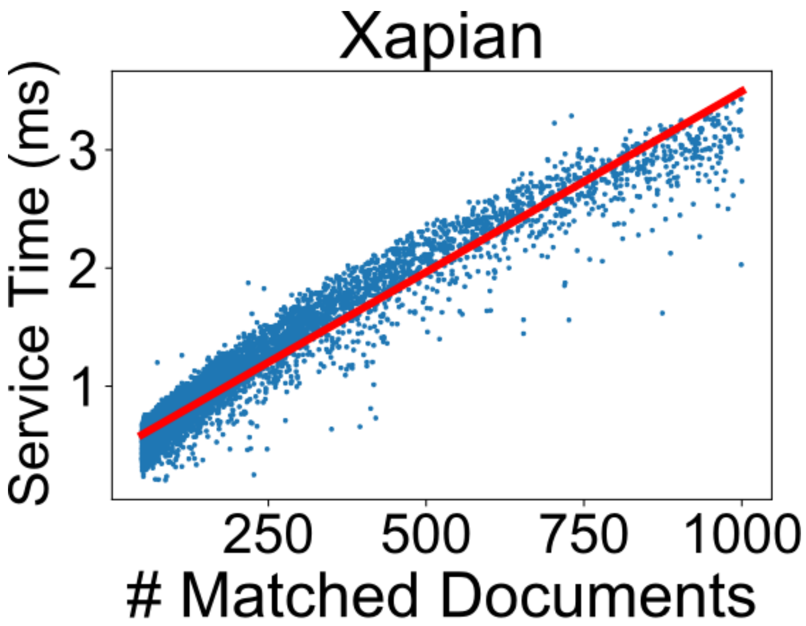
- Real-time translation
- Input request:
  a Spanish phrase

- Speech recognition
- Input request:
  a path to an audio file

CHARACTERIZATION – REQUEST TYPE



- Database (disk/in-memory)
- Input request: TPCC
- ORDER_STATUS and PAYMENT have little-to-no variation
- NEW_ORDER and STOCK_LEVEL require further investigation



- Web search
- Input: a search term

CHARACTERIZATION - SUMMARY

- All the applications have *intuitive* features that correlate strongly with request service time

Characterization - Summary

- All the applications have ***intuitive*** features that correlate strongly with request service time
- The correlation relationship is very simple

Characterization - Summary

- All the applications have ***intuitive*** features that correlate strongly with request service time
- The correlation relationship is very simple
- Classify applications into four categories
  - **Little-to-no-variation:** ImgDNN, Masstree
  - **Predicted by request features:** Moses, Sphinx
  - **Predicted by application features:** Xapian
  - **Combination:** Shore, Silo

CHARACTERIZATION - SUMMARY

- All the applications have ***intuitive*** features that correlate strongly with request service time
- The correlation relationship is very simple

- Classify applications into four categories
  - **Little-to-no-variation:** ImgDNN, Masstree
  - **Predicted by request features:** Moses, Sphinx
  - **Predicted by application features:** Xapian
  - **Combination:** Shore, Silo

We can build a ***simple and effective*** latency prediction model for a ***general*** LC application!

- ReTail: Request-level Latency Prediction to Reduce Tail Latency
- QoS-aware power management for LC apps with request-level latency prediction

- **ReTail feature selection**
  - Selects the features that best correlate with request service time
  - General to any LC application
- **ReTail latency prediction**
  - Linear regression
- **ReTail QoS-aware power management**
  - Decides the best frequency for each request

ReTail Feature Selection

- **Input: a log with**
  - User-provided-set of N samples
  - A menu of features for each request sample
    - » Request features such as request type, request size, etc
    - » Potential intermediate variables in the application
      - – Leverage tracing and logging statements in the source code

ReTail Feature Selection

- **Input: a log with**
  - User-provided-set of N samples
  - A menu of features for each request sample
    - Request features such as request type, request size, etc
    - Potential intermediate variables in the application
      - Leverage tracing and logging statements in the source code
- **Output: the best features that correlate the most with request service time**

RETAIL FEATURE SELECTION

- **Input: a log with**
  - User-provided-set of N samples
  - A menu of features for each request sample
    - » Request features such as request type, request size, etc
    - » Potential intermediate variables in the application
      - – Leverage tracing and logging statements in the source code
- **Output: the best features that correlate the most with request service time**
- **Selection procedure:**
  - Sort all the features in decreasing order of their ***correlation degree***
    - » Numerical feature: Pearson correlation coefficient
    - » Categorical feature: the square of correlation ratio
  - Select the first feature
  - Select one more feature at a time until correlation degree doesn't improve thereafter

CSL

		Model Info				Overhead		Accuracy		
		#Layer	#Neuron/layer	#Epoch	Batch size	Training	Inference	R^2	RMSE	RMSE/QoS
Xapian	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.959	$0.334ms$	4.18%
	NN-Gemini	5	128	15	32	$9.7s$	$363\mu s$	0.973	$0.270ms$	3.38%
	NN-Tuned	1	16	5	32	$0.98s$	$107\mu s$	0.974	$0.264ms$	3.30%
Moses	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.854	$3.622ms$	3.02%
	NN-Gemini	5	128	500	32	$85.1s$	$514\mu s$	0.833	$3.867ms$	3.22%
	NN-Tuned	1	4	400	1024	$0.74s$	$258\mu s$	0.854	$3.617ms$	3.01%
Sphinx	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.746	$217.929ms$	5.45%
	NN-Gemini	5	128	1000	32	$36.15s$	$344\mu s$	0.747	$217.396ms$	5.43%
	NN-Tuned	3	128	700	32	$15.39s$	$300\mu s$	0.747	$217.474ms$	5.43%

- **Categorization and Linear regression**
  - Most relationships are categorical or linear

		Model Info				Overhead		Accuracy		
		#Layer	#Neuron/layer	#Epoch	Batch size	Training	Inference	R^2	RMSE	RMSE/QoS
Xapian	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.959	$0.334ms$	4.18%
	NN-Gemini	5	128	15	32	$9.7s$	$363\mu s$	0.973	$0.270ms$	3.38%
	NN-Tuned	1	16	5	32	$0.98s$	$107\mu s$	0.974	$0.264ms$	3.30%
Moses	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.854	$3.622ms$	3.02%
	NN-Gemini	5	128	500	32	$85.1s$	$514\mu s$	0.833	$3.867ms$	3.22%
	NN-Tuned	1	4	400	1024	$0.74s$	$258\mu s$	0.854	$3.617ms$	3.01%
Sphinx	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.746	$217.929ms$	5.45%
	NN-Gemini	5	128	1000	32	$36.15s$	$344\mu s$	0.747	$217.396ms$	5.43%
	NN-Tuned	3	128	700	32	$15.39s$	$300\mu s$	0.747	$217.474ms$	5.43%

- **Categorization and Linear regression**
  - Most relationships are categorical or linear
  - Comparison with neural networks

ReTail Latency Prediction

		Model Info				Overhead		Accuracy		
		#Layer	#Neuron/layer	#Epoch	Batch size	Training	Inference	R^2	RMSE	RMSE/QoS
Xapian	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.959	$0.334ms$	4.18%
	NN-Gemini	5	128	15	32	$9.7s$	$363\mu s$	0.973	$0.270ms$	3.38%
	NN-Tuned	1	16	5	32	$0.98s$	$107\mu s$	0.974	$0.264ms$	3.30%
Moses	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.854	$3.622ms$	3.02%
	NN-Gemini	5	128	500	32	$85.1s$	$514\mu s$	0.833	$3.867ms$	3.22%
	NN-Tuned	1	4	400	1024	$0.74s$	$258\mu s$	0.854	$3.617ms$	3.01%
Sphinx	Linear Regression	N.A.				$0.003s$	$5\mu s$	0.746	$217.929ms$	5.45%
	NN-Gemini	5	128	1000	32	$36.15s$	$344\mu s$	0.747	$217.396ms$	5.43%
	NN-Tuned	3	128	700	32	$15.39s$	$300\mu s$	0.747	$217.474ms$	5.43%

- **Categorization and Linear regression**
  - Most relationships are categorical or linear
  - Comparison with neural networks
    - Small training and inference overhead

ReTail Latency Prediction

		Model Info				Overhead		Accuracy		
		#Layer	#Neuron/layer	#Epoch	Batch size	Training	Inference	R^2	RMSE	RMSE/QoS
Xapian	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.959	$0.334ms$	4.18%
	NN-Gemini	5	128	15	32	$9.7s$	$363\mu s$	0.973	$0.270ms$	3.38%
	NN-Tuned	1	16	5	32	$0.98s$	$107\mu s$	0.974	$0.264ms$	3.30%
Moses	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.854	$3.622ms$	3.02%
	NN-Gemini	5	128	500	32	$85.1s$	$514\mu s$	0.833	$3.867ms$	3.22%
	NN-Tuned	1	4	400	1024	$0.74s$	$258\mu s$	0.854	$3.617ms$	3.01%
Sphinx	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.746	$217.929ms$	5.45%
	NN-Gemini	5	128	1000	32	$36.15s$	$344\mu s$	0.747	$217.396ms$	5.43%
	NN-Tuned	3	128	700	32	$15.39s$	$300\mu s$	0.747	$217.474ms$	5.43%

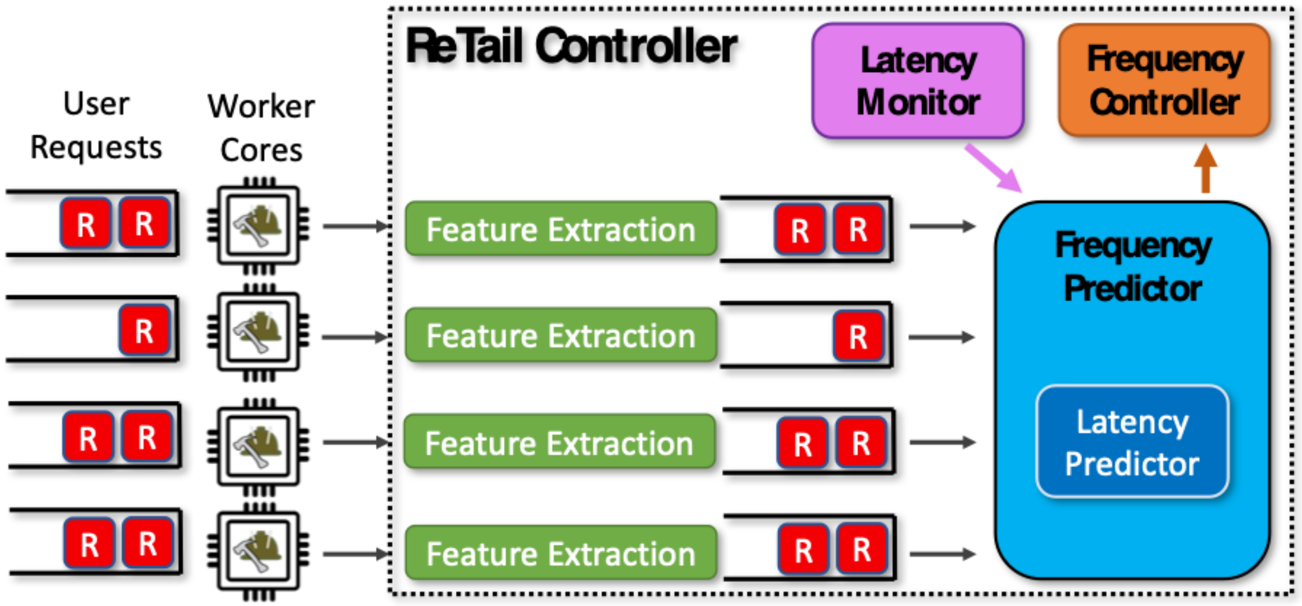
- **Categorization and Linear regression**
  - Most relationships are categorical or linear
  - Comparison with neural networks
    - Small training and inference overhead
    - Nearly the same accuracy as neural network

ReTail Latency Prediction

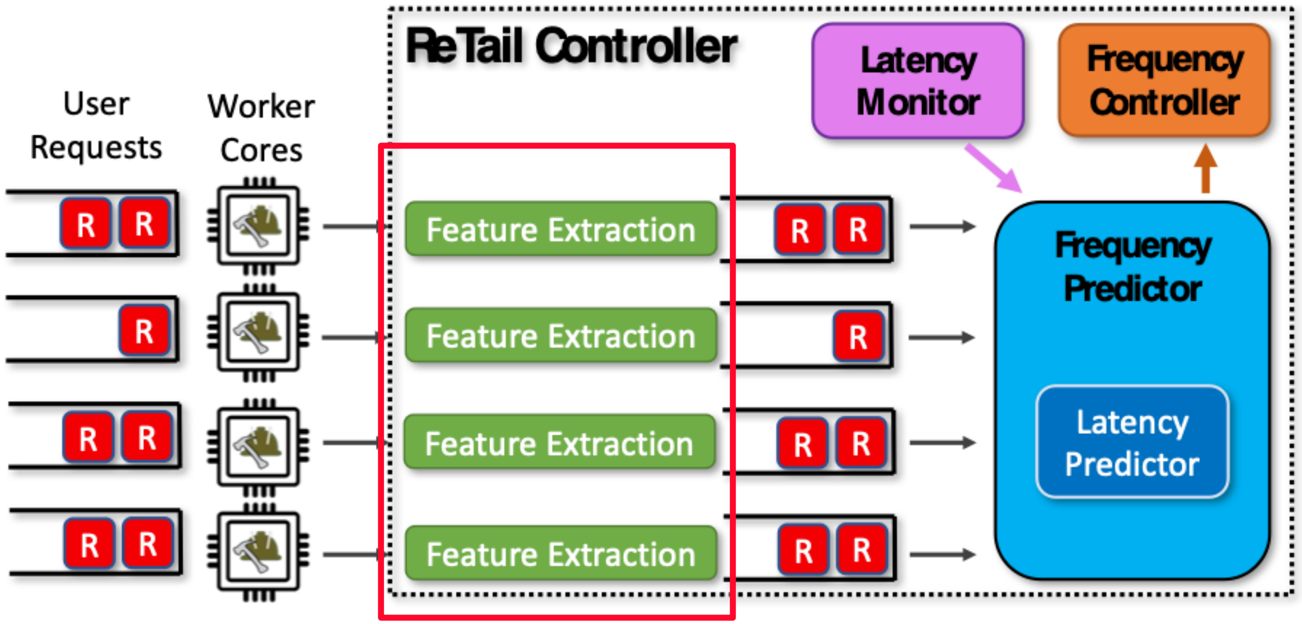
		Model Info				Overhead		Accuracy		
		#Layer	**#Neuron/layer**	**#Epoch**	**Batch size**	**Training**	**Inference**	R^2	**RMSE**	**RMSE/QoS**
Xapian	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.959	$0.334ms$	4.18%
	NN-Gemini	5	128	15	32	$9.7s$	$363\mu s$	0.973	$0.270ms$	3.38%
	NN-Tuned	1	16	5	32	$0.98s$	$107\mu s$	0.974	$0.264ms$	3.30%
Moses	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.854	$3.622ms$	3.02%
	NN-Gemini	5	128	500	32	$85.1s$	$514\mu s$	0.833	$3.867ms$	3.22%
	NN-Tuned	1	4	400	1024	$0.74s$	$258\mu s$	0.854	$3.617ms$	3.01%
Sphinx	**Linear Regression**	N.A.				$0.003s$	$5\mu s$	0.746	$217.929ms$	5.45%
	NN-Gemini	5	128	1000	32	$36.15s$	$344\mu s$	0.747	$217.396ms$	5.43%
	NN-Tuned	3	128	700	32	$15.39s$	$300\mu s$	0.747	$217.474ms$	5.43%

- **Categorization and Linear regression**
  - Most relationships are categorical or linear
  - Comparison with neural networks
    - Small training and inference overhead
    - Nearly the same accuracy as neural network
  - Explainable

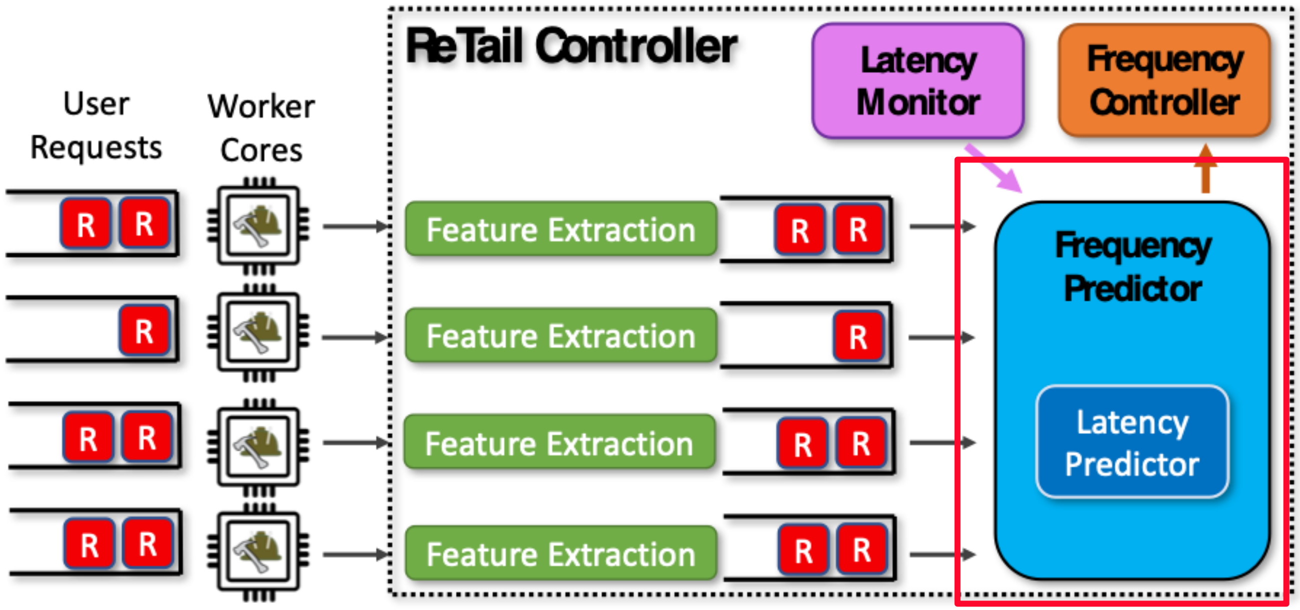
- Find the minimum frequency to satisfy QoS



- Find the minimum frequency to satisfy QoS



- Find the minimum frequency to satisfy QoS

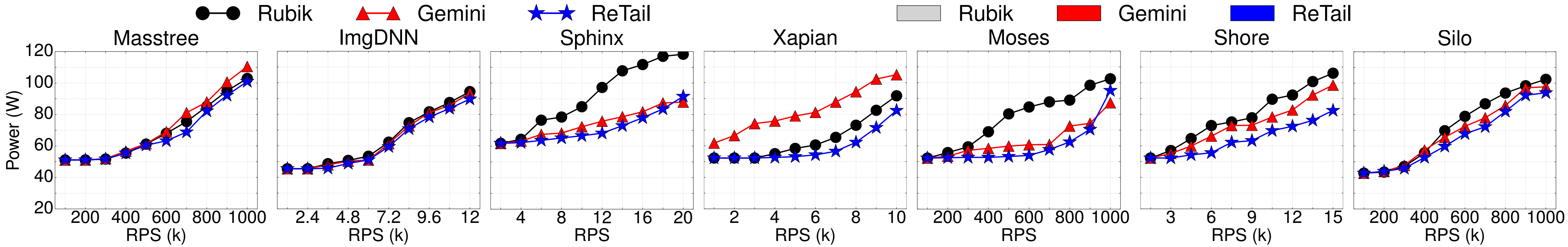


- **ReTail feature selection**
  - Timeliness of all the selected features
  - Correlation degree of multiple features
- **ReTail latency prediction**
  - Training datasets
  - Model retraining for model drift
- **ReTail power management**
  - Prediction based on all queued and newly joined requests
  - Feedback-control loop with latency monitoring

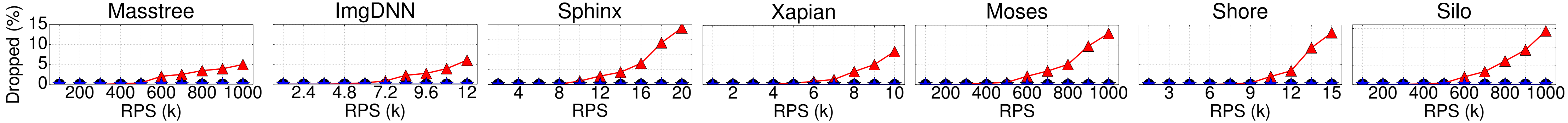
EVALUATION - METHODOLOGY

- Server: Intel Xeon Gold 6152 CPU @ 2.1GHz
  - Power manager: one reserved core in socket 0
  - LC app: socket 0
  - Clients: socket 1
- Power measurement: CPU Energy Meter
  - Measures energy consumption of socket 0
  - Divides the execution time of the LC app
- ACPI-Freq: 1~2.1GHz in 0.1GHz steps
- Baselines:
  - **Rubik [MICRO'15]**: statistical model
  - **Gemini [MICRO'20]**: NN-based, only considers request features

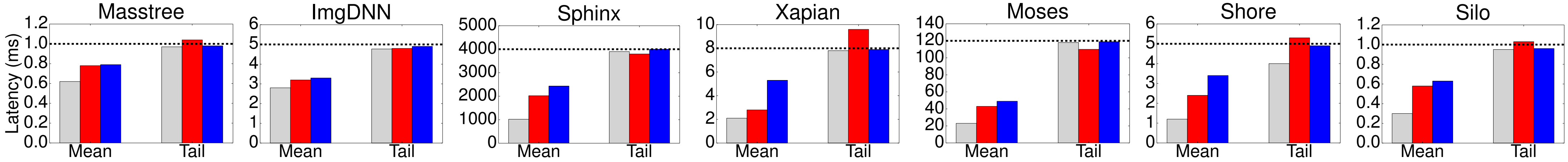
EVALUATION



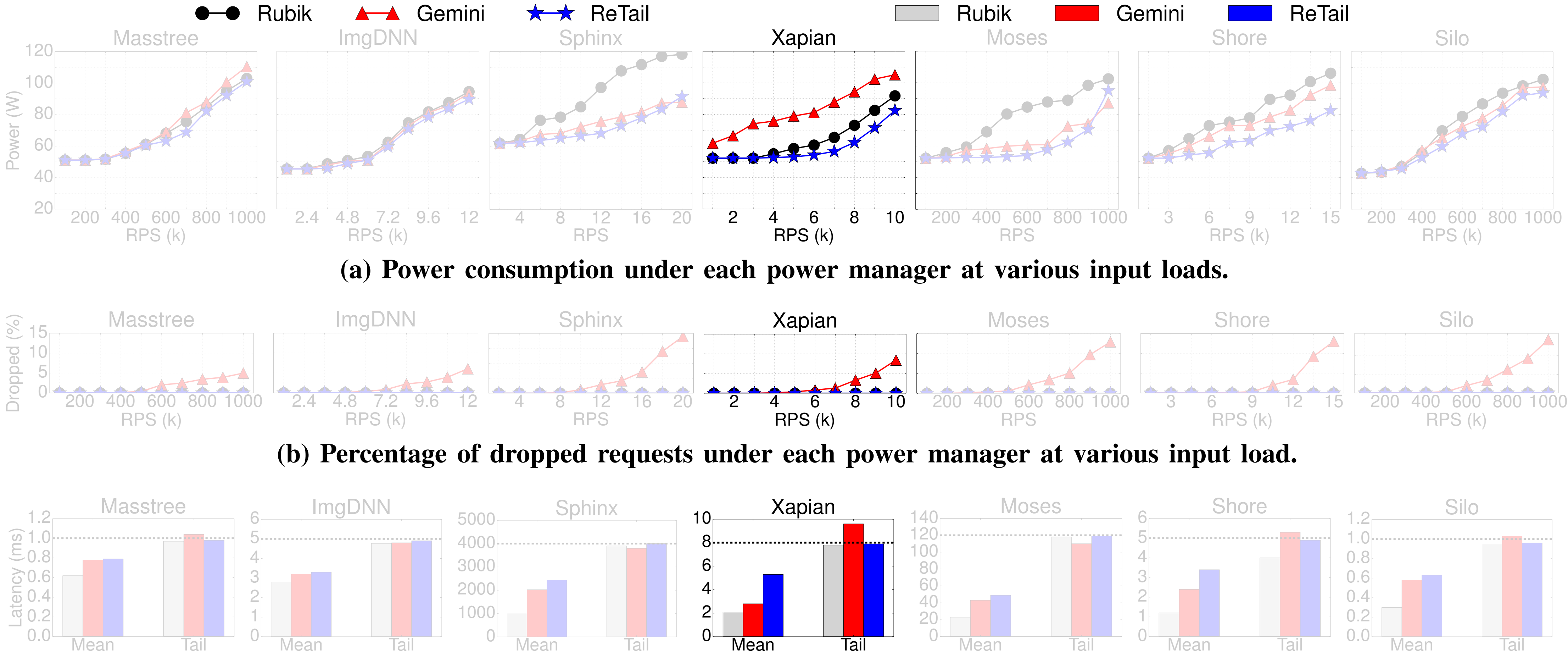
(a) Power consumption under each power manager at various input loads.



(b) Percentage of dropped requests under each power manager at various input load.



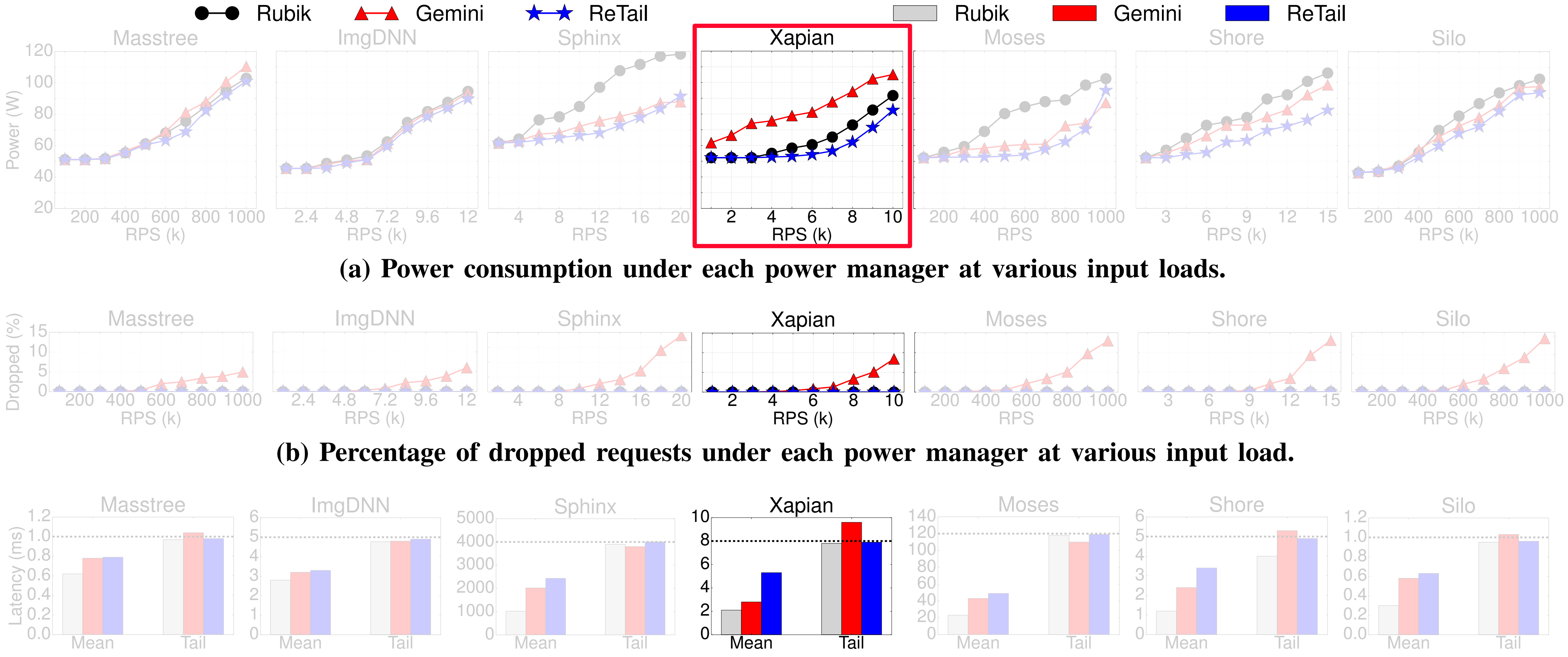
(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.



(a) Power consumption under each power manager at various input loads.

(b) Percentage of dropped requests under each power manager at various input load.

(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

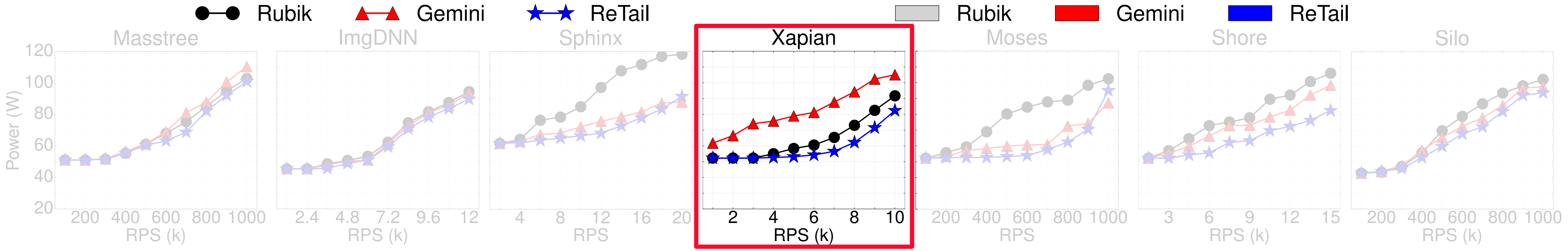


(a) Power consumption under each power manager at various input loads.

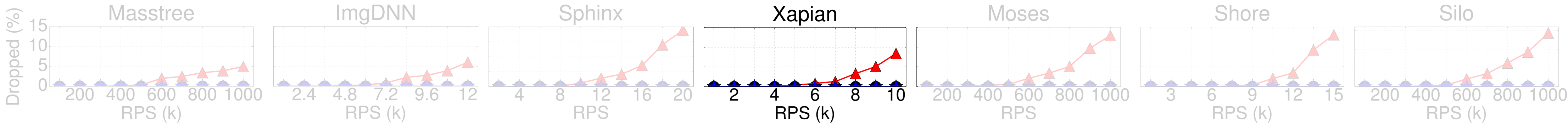
(b) Percentage of dropped requests under each power manager at various input load.

(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

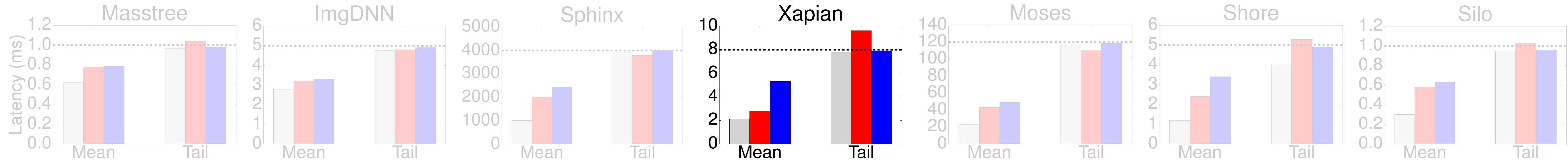
EVALUATION



(a) Power consumption under each power manager at various input loads.

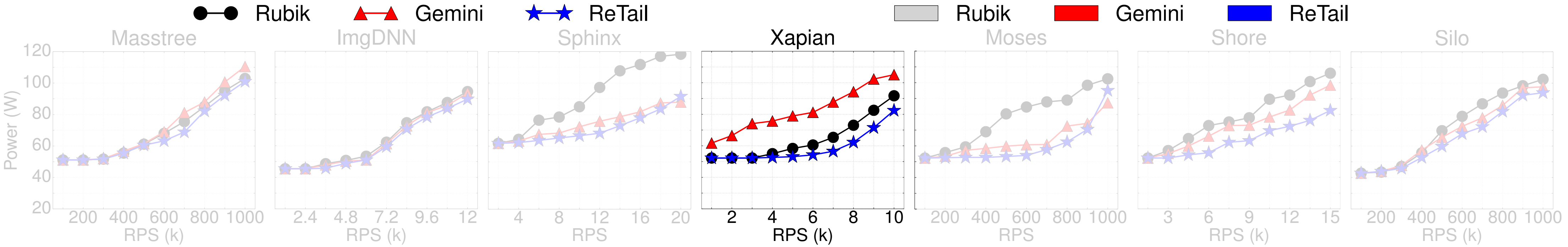


(b) Percentage of dropped requests under each power manager at various input load.

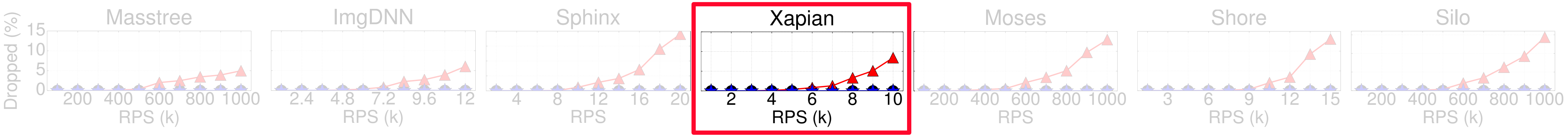


(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

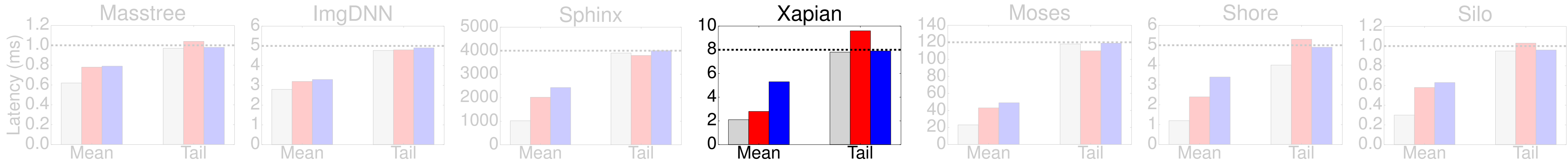
- 12% and 9% power saving compared to Rubik and Gemini, respectively



(a) Power consumption under each power manager at various input loads.



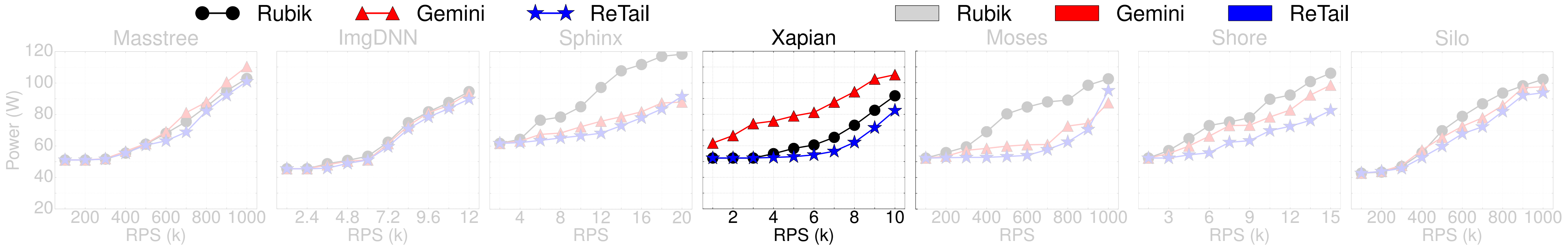
(b) Percentage of dropped requests under each power manager at various input load.



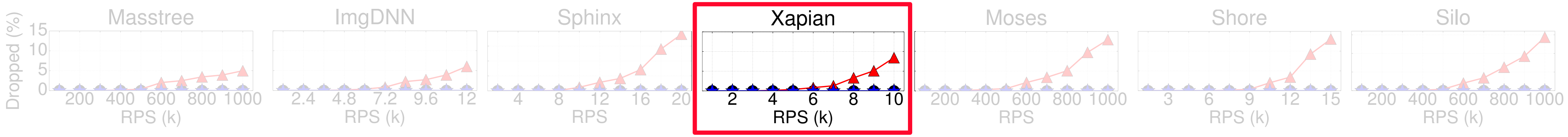
(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

- 12% and 9% power saving compared to Rubik and Gemini, respectively

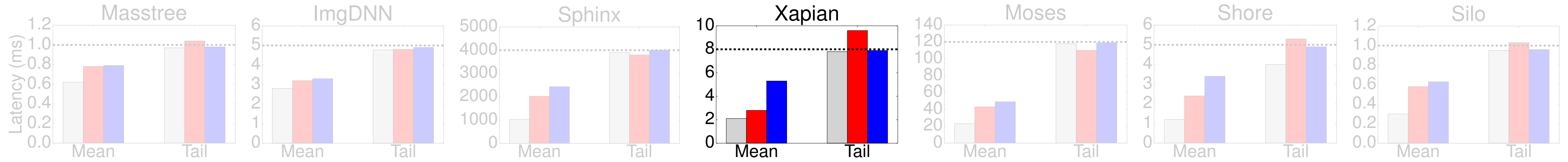
EVALUATION



(a) Power consumption under each power manager at various input loads.

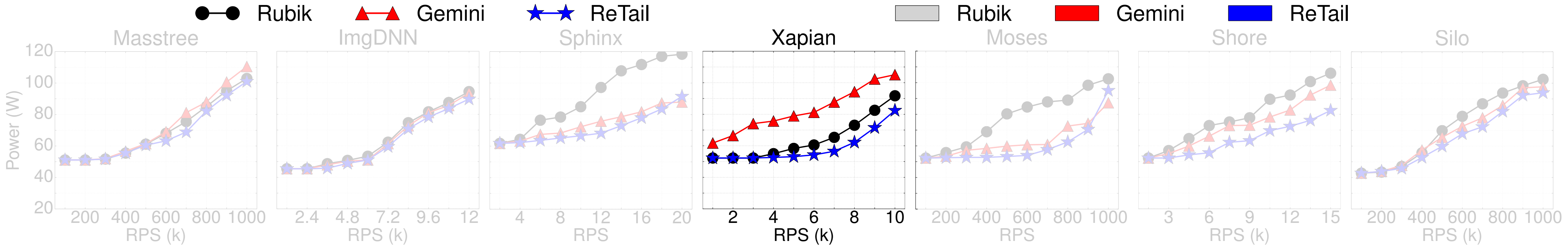


(b) Percentage of dropped requests under each power manager at various input load.

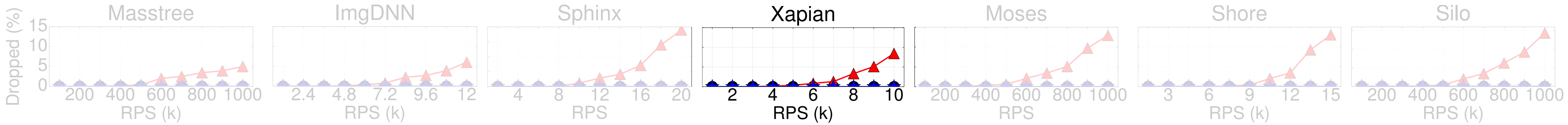


(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

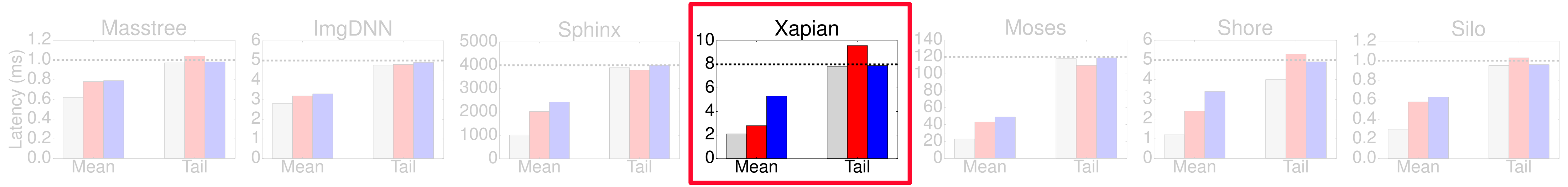
- 12% and 9% power saving compared to Rubik and Gemini, respectively
- No dropping of any requests



(a) Power consumption under each power manager at various input loads.

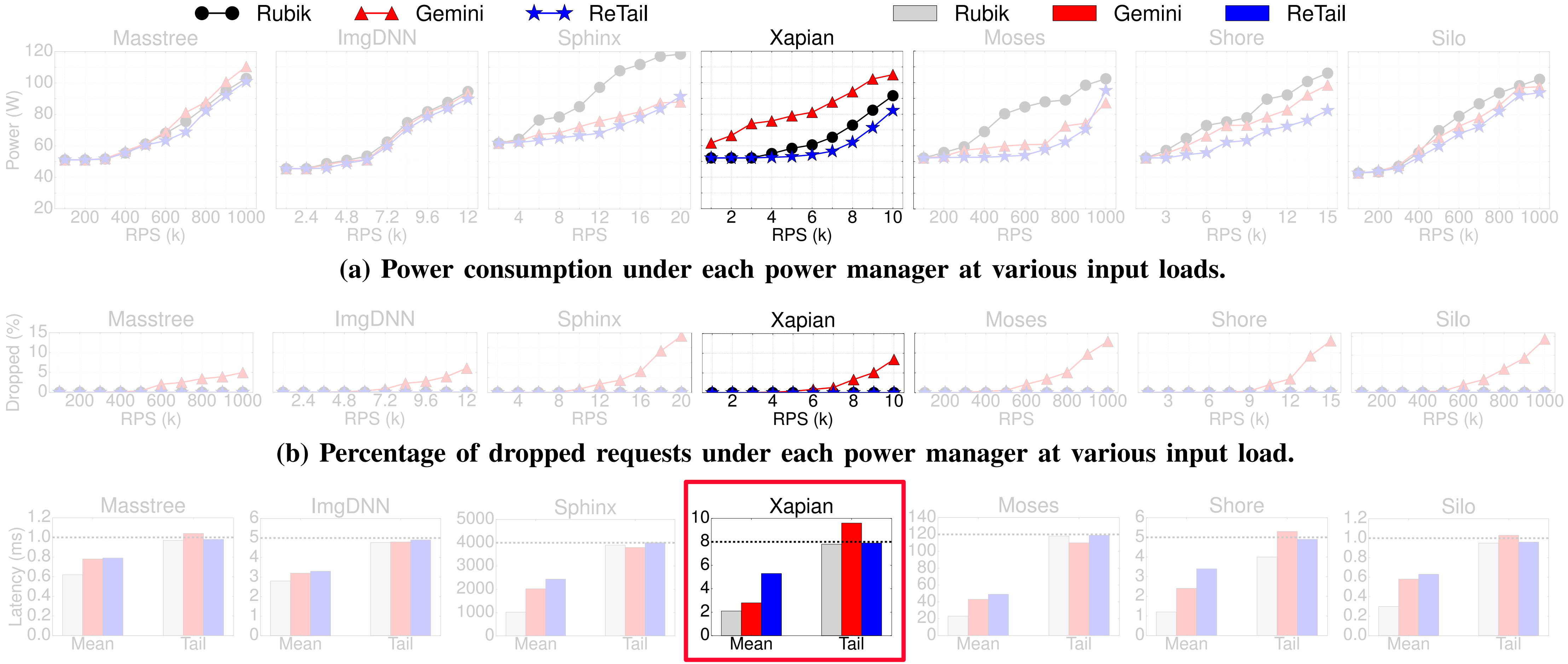


(b) Percentage of dropped requests under each power manager at various input load.



(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

- 12% and 9% power saving compared to Rubik and Gemini, respectively
- No dropping of any requests



(a) Power consumption under each power manager at various input loads.

(b) Percentage of dropped requests under each power manager at various input load.

(c) Mean and tail latency under each power manager at max load. The horizontal dotted lines are the QoS targets.

- 12% and 9% power saving compared to Rubik and Gemini, respectively
- No dropping of any requests
- Meet QoS

	Masstree	ImgDNN	Sphinx	Xapian	Moses	Shore	Silo
Rubik	0.05	0.9	2500	2.8	47.1	3.9	0.5
Gemini	0.03	0.8	217	3.6	3.6	2.2	0.2
ReTail	0.04	0.8	217	0.3	3.6	0.3	0.1

- **ReTail has the lowest Root-Mean-Square-Error (RMSE)**
- **ReTail outperforms Gemini's more sophisticated NN model because**
  - NN's high inference overhead delays frequency adjustments
  - Gemini only considers request features, while ReTail also considers application features

- Leveraging request-level latency prediction to improve power efficiency

- ReTail feature selection
- ReTail latency prediction: a simple learning model is good enough!!
- ReTail power management

- Power saving up to 36% (average 9%) compared to the best state-of-the-art power manager without QoS violations

- Future work: many potential uses of the prediction model!

Cornell University
Computer Systems Laboratory

RETAIL:
OPTING FOR LEARNING SIMPLICITY TO ENABLE QOS-AWARE POWER MANAGEMENT IN THE CLOUD

Thanks!

Offline discussion: chenshuang0804@gmail.com