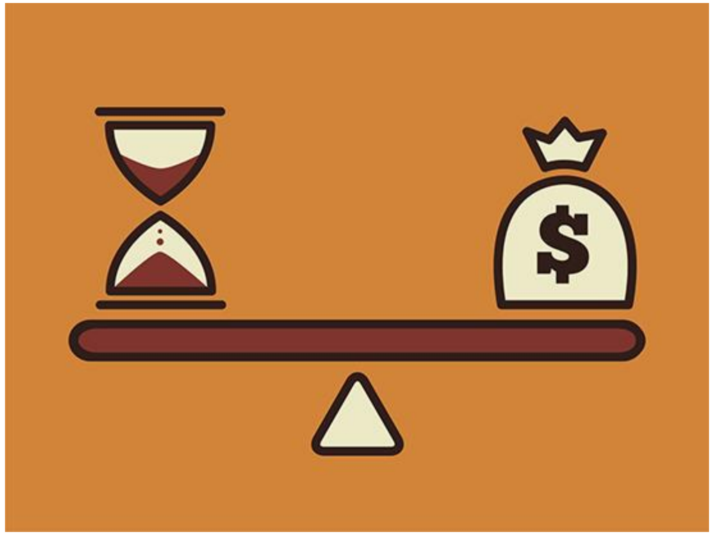
CLAMShell: Speeding up Crowds for Low-latency Data Labeling

Daniel Haas, Jiannan Wang, Eugene Wu, Michael J. Franklin

Crowd Latency in Data Labeling

- Necessary to use crowdsourcing method for data labeling
- Desire: low cost, high speed, high quality
- Trade-off between cost and latency for crowd-sourced labeling tasks.



CLAMShell System

- speeds up crowds in order to achieve consistent, low-latency data labeling
- a collection of practical techniques
- reduces latency in all stages of labeling tasks

Contribution

- An empirical study of the dominant sources of latency
- CLAMShell: systematically provide solutions for each major sources of latencies
- Evaluation of CLAMShell on live workers

Study Crowd Latency - Sources

- Categorizing the factors based on the granularity of work

1. Per-Task Latency
2. Per-Batch Latency
3. Full-Run Latency

Sources of Latency

1. Per-Task Latency
   - Recruitment: recruiting the crowd workers
   - Qualification and Training: tutorials or qualification tasks
   - Work: workers’ status may be very different
2. Per-Batch Latency
3. Full-Run Latency

Sources of Latency

1. Per-Task Latency
2. Per-Batch Latency

   Batch: labeling tasks in fixed-sized set

   Latency distribution and long tails

   - Stragglers: the batch must block until the slowest task is completed
   - Mean Pool Latency (MPL)
   - Pool and Worker Variance: high variance within and between batches

3. Full-Run Latency

Sources of Latency

1. Per-Task Latency
2. Per-Batch Latency
3. Full-Run Latency
   - Decision Latency: pick next batches
   - Task Count: machine learning
   - Batch Size
   - Pool Size

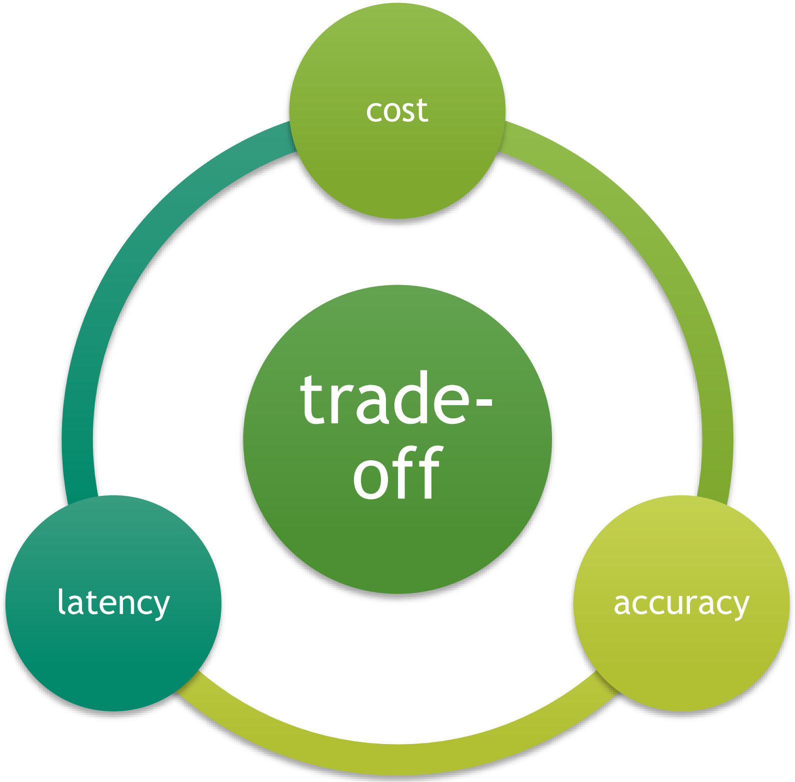
Sources of Latency

Task Latency	Batch Latency	Full-Run Latency
Recruitment	Stragglers	Decision Time
Qual & Training	Mean pool latency	Task Count
Work	Pool variance	Batch Size
		Pool Size

Existing Solutions and Researches

- frequently repost tasks: high recruitment time
- algorithmically increase prices over time to attract more workers
- retainer model: pre-recruits a pool of crowd workers
- re-designing task interfaces: task specific
- using algorithmic analysis and machine learning to reduce task count
  - Active learning: using data from completed tasks until the prediction quality exceeds a user-defined threshold
  - Batch size limitation

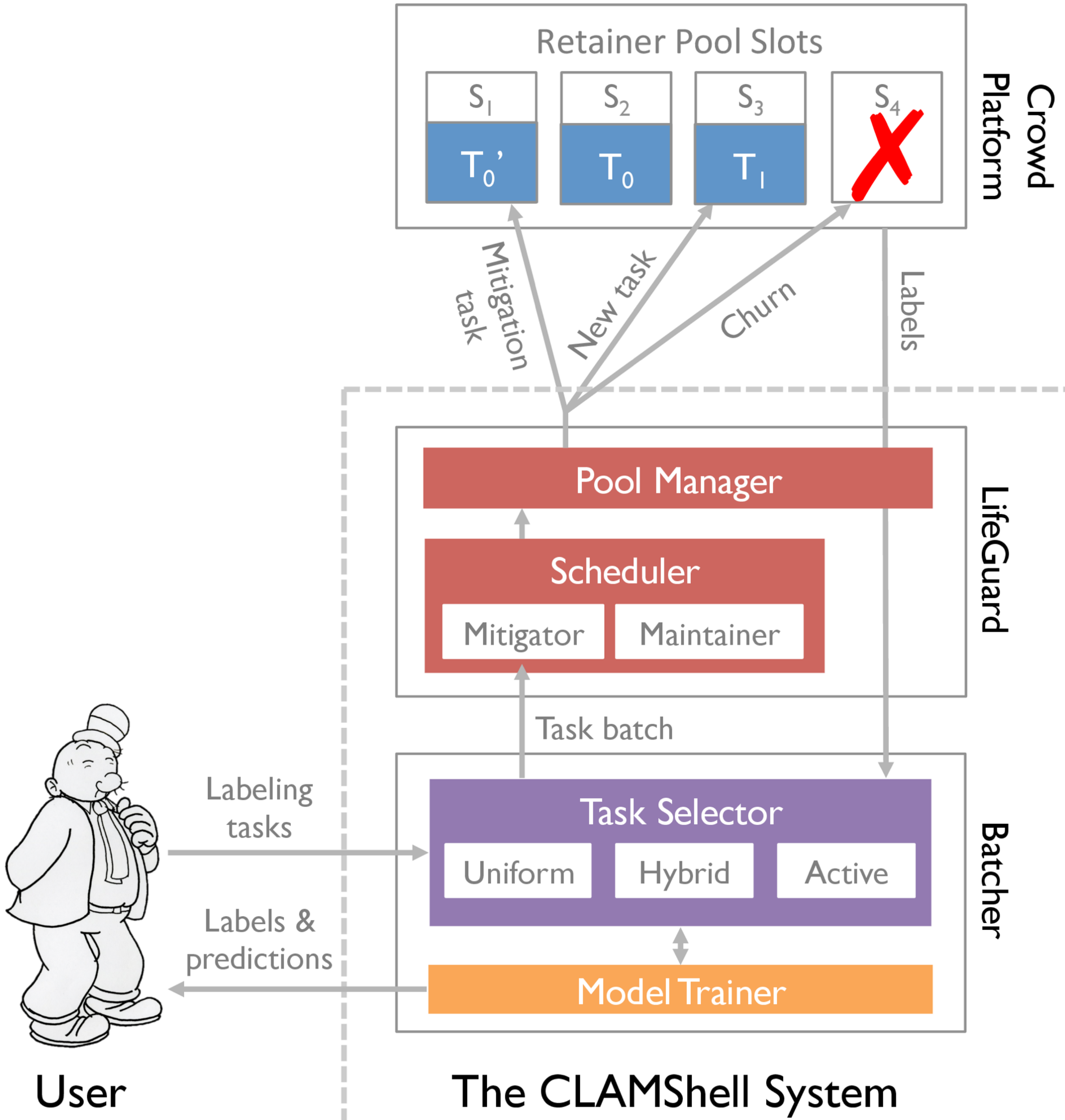
Reducing Latency – Our Thought



- Our Solution: CLAMShell
- reducing latency by sacrificing cost
- comprehensive solution
- general purpose labeling system

CLAMShell System

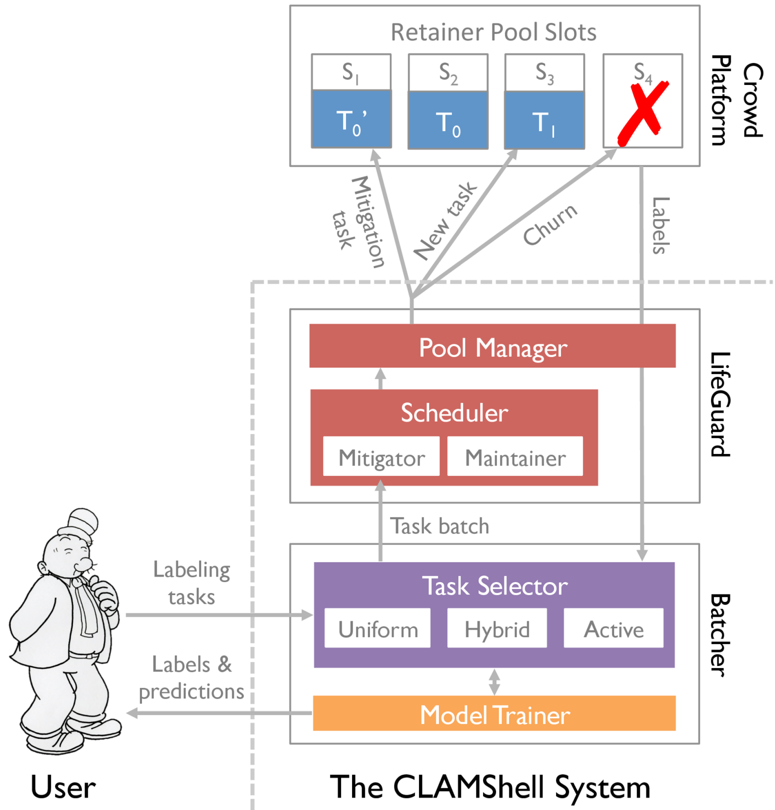
1. Task Latency
   - Retainer pools
   - Includes workers training and qualification in recruitment
2. Batch Latency
   - Straggler mitigation
   - Pool maintenance
3. Full-Run Latency
   - Hybrid strategy: active learning + passive learning



Retainer Pool Slots
S_1
S_2
S_3
S_4
T_0'
T_0
T_1
Crowd Platform
Mitigation task
New task
Churn
Labels
Pool Manager
Scheduler
Mitigator
Maintainer
LifeGuard
Task batch
Labeling tasks
Labels & predictions
Task Selector
Uniform
Hybrid
Active
Model Trainer
Batcher
User
The CLAMShell System

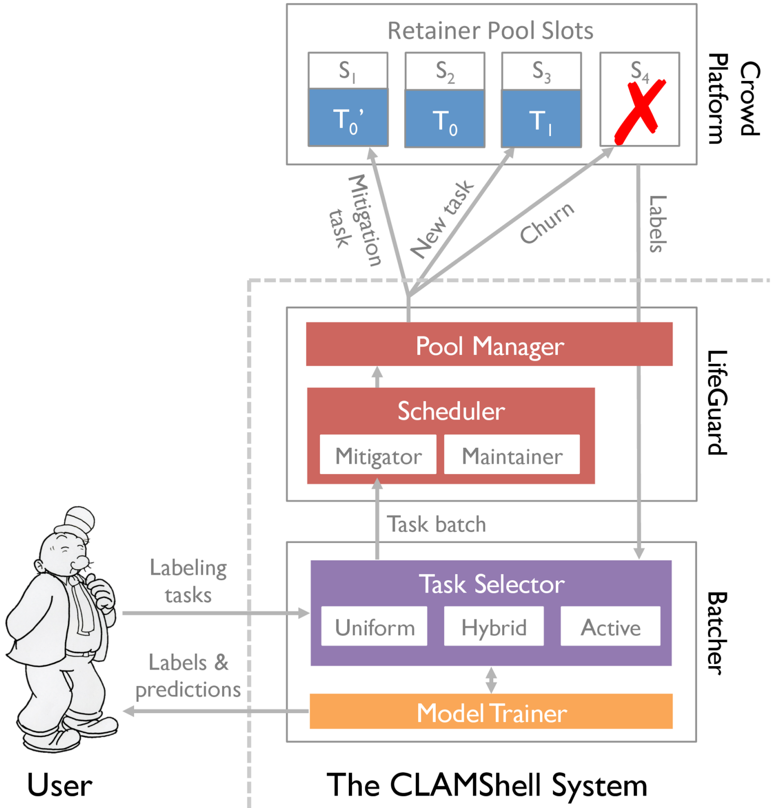
CLAMShell System - Architecture

- User submits labeling tasks to **Batcher**
- **Task Selector** picks incomplete tasks and sends to **LifeGuard**
- **LifeGuard** schedules tasks in batches and sends to **Crowd Platform**



CLAMShell System - Architecture

- **Crowd Platform**
  - Slots: retainer tasks
    - empty, new task or duplicated task
  - Completed labels are sent back to **Batcher**
- Machine learning model: hybrid sampler
- User can access the completed labels and query for new predictions



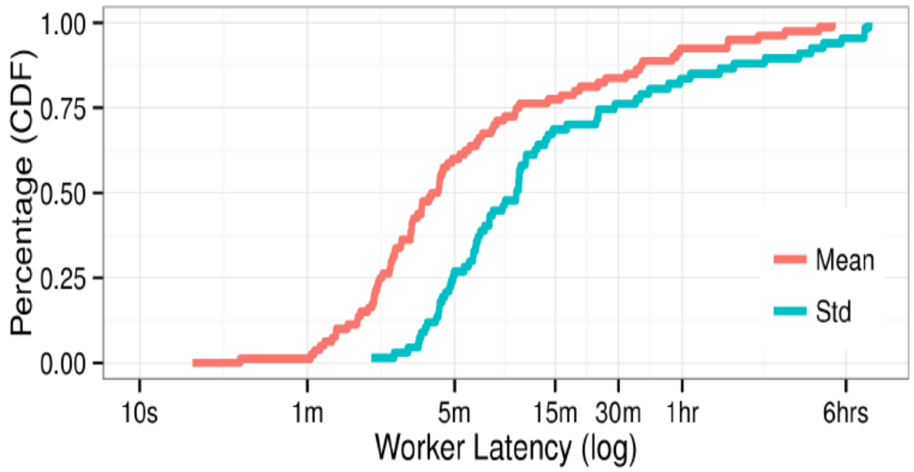
CLAMShell System - Optimization

Task Latency	Batch Latency	Full-Run Latency
Recruitment	Stragglers	Decision Time
Qual & Training	Mean pool latency	Task Count
Work	Pool variance	Batch Size
		Pool Size

1. Task Latency
   - Retainer pools
   - Includes workers training and qualification in recruitment
2. Batch Latency
   - Straggler mitigation
   - Pool maintenance
3. Full-Run Latency
   - Hybrid strategy: active learning + passive learning

Batch Latency Optimization

- variability of worker latencies within the pool
- variability within the tasks that a single worker performs
- reduce both the mean of the latency distribution and its variance
- Straggler Mitigation and Pool Maintenance



Straggler Mitigation: Reducing Variance

- replication-based approach
  - worker: active / available
  - task: active / complete / unassigned
- Default: route unassigned tasks to available workers
  - Batch is finished until the slowest task completed
- Straggler mitigation: available workers received duplication of active tasks immediately
  - User gets the first completed copy and other copies get terminated
  - Hide latency by sending task to other workers

Straggler Mitigation - Simulation

- Q1: Which task should be assigned to an available worker?
  - longest-running active task, random task, task with fewest active workers or task known by an oracle to complete the slowest
- Simulation result: the selection result doesn't affect end-to-end latency.
  - random performed as fast as the oracle solution
  - fast workers complete almost all of the tasks

Straggler Mitigation - Simulation

- Q2: What is the most effective batch size for Straggler Mitigation?
  - Let pool size to batch size ratio $R = \frac{N_{pool}}{N_{batch}}$
- Simulation result
  - Using random selection algorithm and different pool size and R ratio
  - Each batch gains more benefit from Straggler Mitigation when R is higher

Pool Maintenance: Better Mean Latency

- The workers in labeling pool are slow on average
- Strategy: continuously replaces slow workers in order to converge to a pool of mostly fast workers.
- Latency threshold PM_ℓ
- Calculate mean latency for each worker based on finished task
- Reserve new workers in background for replacement

Pool Maintenance - Speed Convergence

- Mean latencies for a global set of workers: μ_i
  - $\mu_f < PM_\ell$ mean latency among fast workers with probability $1 - q$
  - $\mu_s > PM_\ell$ mean latency among slow workers with probability q
- Mean latency:
  - Initial: $\mathbb{E}[\mu_i] = (1 - q)\mu_f + q\mu_s$
  - After first step: $\mathbb{E}[\mu_i] = (1 - q)\mu_f + (q(1 - q)\mu_f + q^2\mu_s)$
  - After nth step:

$$\begin{aligned}\mathbb{E}[\mu_i] &= (\sum_{i=0}^{n} q^i)(1-q)\mu_f + q^{n+1}\mu_s \\ &= (1-q^{n+1})\mu_f + q^{n+1}\mu_s.\end{aligned}$$

$$\lim_{n\to\infty} \mathbb{E}[\mu_i] = \mu_f$$

Pool Maintenance

- Simulation: replace slow workers after each batch
  - batch latency falls quickly, nearly halving in just 15 to 20 batches
  - converges quickly to the model’s predicted asymptote
- Threshold Selection: k standard deviations below the mean
  - low enough to decrease average pool latency by releasing slow workers
  - high enough to avoid discarding the fastest workers from the pool
- Pool Maintenance can be use in other critiria
  - quality
  - weighted average of quality and speed

Batch Latency - Combination

- Naïve approach
  - Simply combine Straggler Mitigation and Pool Maintenance together
  - Result: zero or negative gains compare to Straggler Mitigation alone
  - Straggler Mitigation terminate slow tasks, skewing the latency of each worker

Batch Latency - Combination

- TermEst
  - estimate the average latencies of terminated tasks based on the number of times a worker's task is terminated and the fast workers latency

$$l_{s,T_t} = \frac{l_f(N+\alpha)}{N_c+\alpha}$$

  - using estimated latencies on terminated tasks to calculate the latencies for slow workers

$$l_s = \frac{N_t}{N} \times l_{s,T_t} + \frac{N_c}{N} \times l_{s,T_c}$$

Batch Latency - Quality Control

- What if fast workers are spammers or inaccurate workers?
- In empirical data, fast workers are no more likely to be inaccurate than slow workers
- Traditional quality control techniques are entirely complementary to our techniques
  - redundancy-based quality control algorithms
    - P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on Amazon Mechanical Turk. SIGKDD, 2010.
    - M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. Science, 2015.

CLAMShell System - Optimization

Task Latency	Batch Latency	Full-Run Latency
Recruitment	Stragglers	Decision Time
Qual & Training	Mean pool latency	Task Count
Work	Pool variance	Batch Size
		Pool Size

1. Task Latency
   - Retainer pools
   - Includes workers training and qualification in recruitment
2. Batch Latency
   - Straggler mitigation
   - Pool maintenance
3. Full-Run Latency
   - Hybrid strategy: active learning + passive learning

Full-Run Latency Optimization

- Learning Algorithm decreases task count, but is restricted by decision latency and batch size
- CLAMShell uses uncertainty sampling to reduce the task count even further
  - Increasing decision latency
  - Decreasing batch size
- Hybrid learning: combines active and passive learning
  - maximize pool parallelism
  - hide batch size limitation

Hybrid learning

- Challenge of active learning
  - A good batch size for learning algorithm to converge
  - It is hard to train a good model on some labeling task
- Hybrid learning
  - simultaneously acquires labels using the active selection strategy and random sampling
  - Point Selection: each worker in the pool has at least one point to label
  - Model Retraining: retrains a model on all previously observed labels, both active and passive learning samples
  - Future work: weight on both types of points can be adjust by user

Hybrid learning – Batch Size

- Small: will take long time to label all points
- Large: slow on training, hard to converge
- According to our experiment, 10 to 40 is the a reasonable range for batch size
- With in that range, there was no significant correlation between batch size and convergence rates on any single dataset

Hybrid learning - Decision Latency

- How to reduce the time to retrain a model?
- First, CLAMShell consider only a uniform random sample of the points for selection in next batch
  - Instead of considering all unlabeled points
- Second, CLAMShell continually retrains models asynchronously on the latest available points
  - There always a new trained module and a new batch available

CLAMShell System - Optimization

CLAMShell Techniques	Latency Mean	Latency Variance	Cost	General
straggler	Yes	Yes	Increase	Yes
pool	Yes	Yes	No Change	Yes
hybrid	Yes	No	Increase	AL

Table 2: CLAMShell techniques (AL: Active Learning).

Evaluation

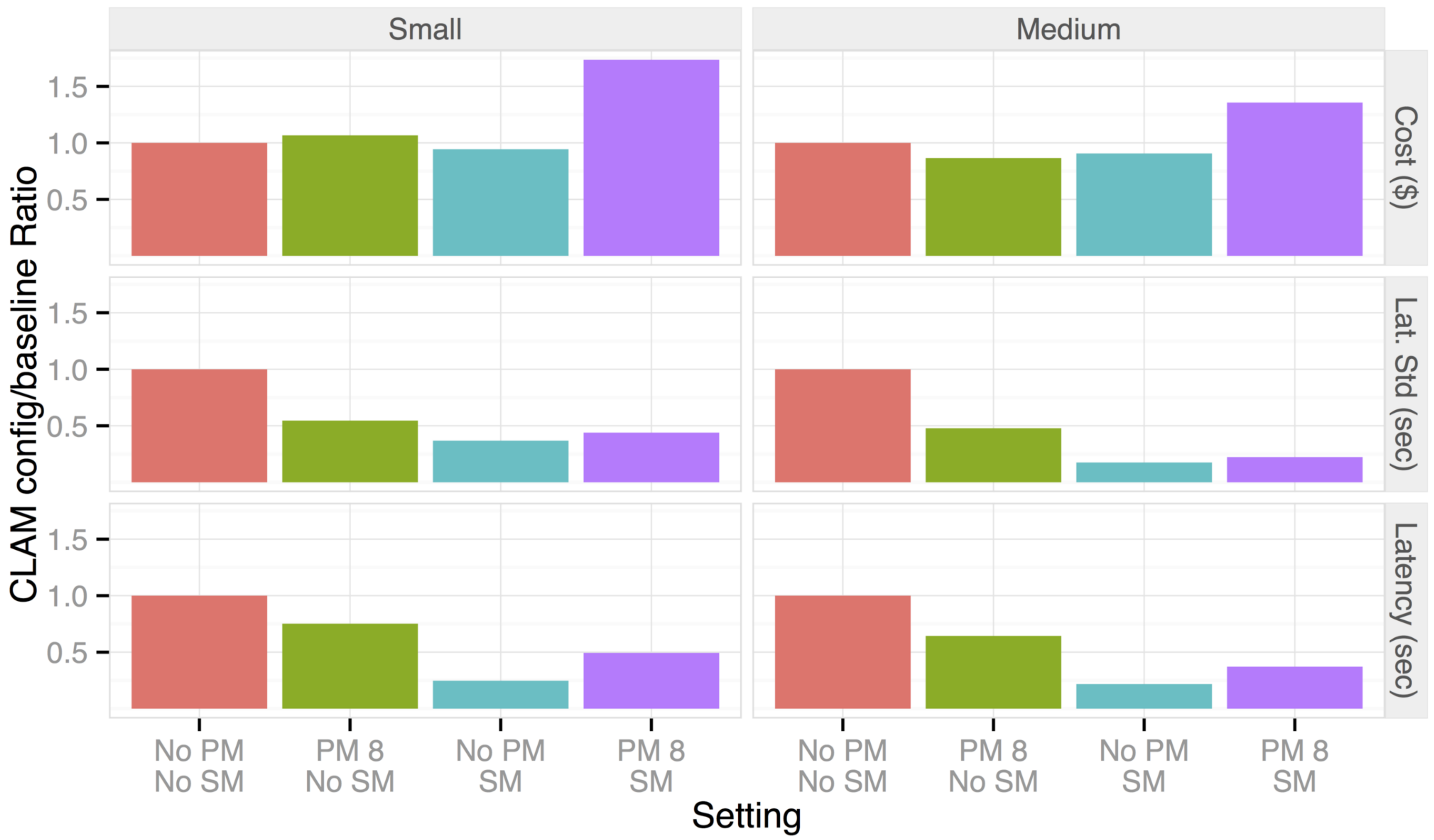
- Simulator:
  - retainer-pool crowd workers
  - uncertainty sampling on top of scikit-learn's model training
- Live Experiments:
  - deploy data labeling task on MTurk
  - run at multiple times of day
  - nearly 250,000 individual task assignments over several weeks

Evaluation – Dataset

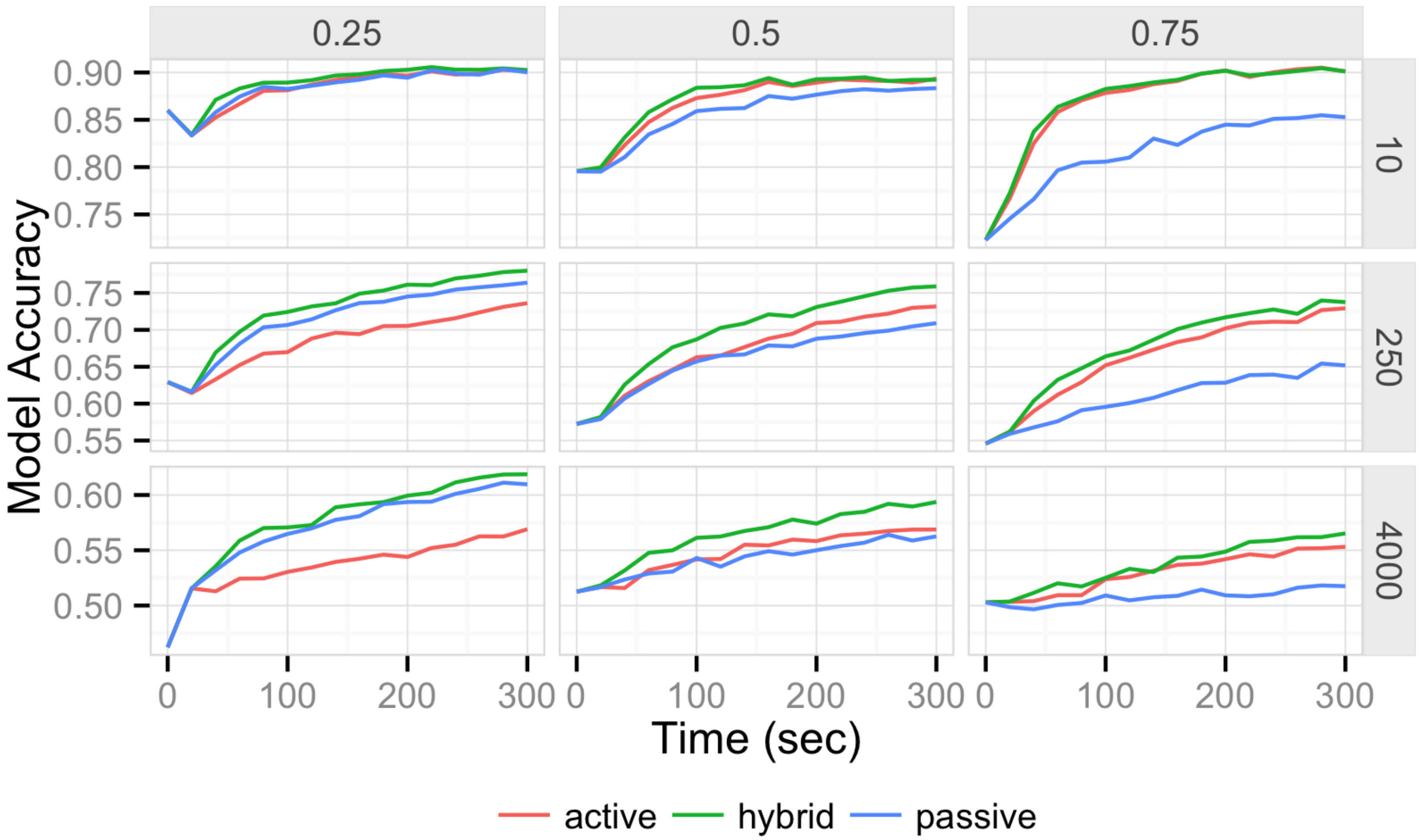
Name	# Instances	Multi-class	Features
MNIST	70,000	Yes	784
CIFAR-10	60,000	2	3072

- Public Dataset
- Machine generated data:
  - scikit-learn data generator

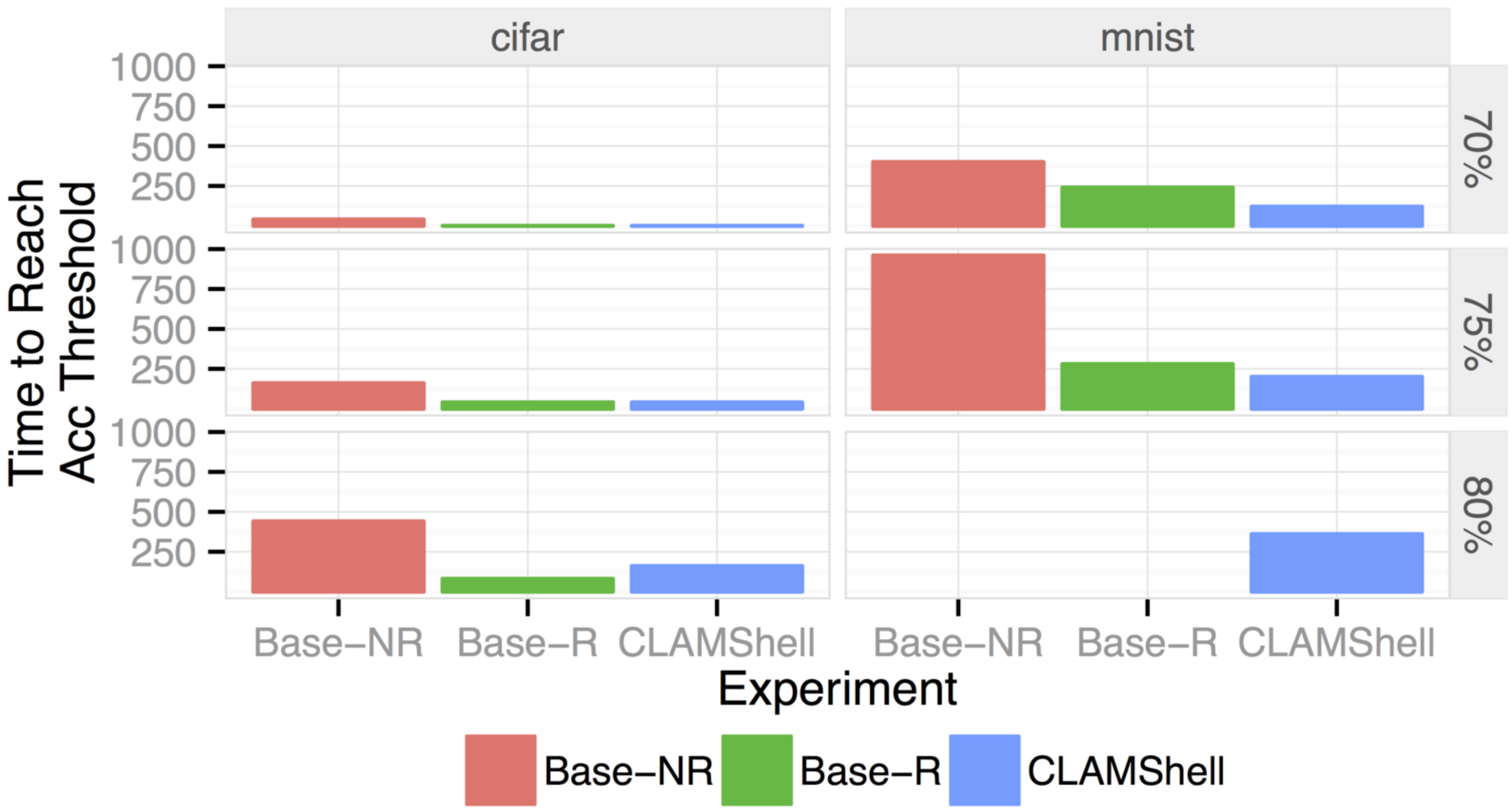
Evaluation – Per-batch Tecniques



Evaluation – Hybrid Learning



Evaluation – Over All



Conclusion

- CLAMShell: Crowdsourcing data labeling system at interactive speeds
  - Straggler mitigation, Pool maintenance and Hybrid learning
- Future work
  - richer objective functions
  - better way to train hybrid learning model
  - integrating CLAMShell with data cleaning system