Distributed Latent Dirichlet Allocation on Streams

YUNYAN GUO, Harbin Institute of Technology, China
JIANZHONG LI, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China and Harbin Institute of Technology, China

Latent Dirichlet Allocation (LDA) has been widely used for topic modeling, with applications spanning various areas such as natural language processing and information retrieval. While LDA on small and static datasets has been extensively studied, several real-world challenges are posed in practical scenarios where datasets are often huge and are gathered in a streaming fashion. As the state-of-the-art LDA algorithm on streams, *Streaming Variational Bayes* (SVB) introduced *Bayesian updating* to provide a streaming procedure. However, the utility of SVB is limited in applications since it ignored three challenges of processing real-world streams: *topic evolution*, *data turbulence*, and *real-time inference*. In this article, we propose a novel distributed LDA algorithm—referred to as *StreamFed-LDA*—to deal with challenges on streams. For topic modeling of streaming data, the ability to capture evolving topics is essential for practical online inference. To achieve this goal, *StreamFed-LDA* is based on a specialized framework that supports lifelong (continual) learning of evolving topics. On the other hand, data turbulence is commonly present in streams due to real-life events. In that case, the design of *StreamFed-LDA* allows the model to learn new characteristics from the most recent data while maintaining the historical information. On massive streaming data, it is difficult and crucial to provide real-time inference results. To increase the throughput and reduce the latency, *StreamFed-LDA* introduces additional techniques that substantially reduce both computation and communication costs in distributed systems. Experiments on four real-world datasets show that the proposed framework achieves significantly better performance of online inference compared with the baselines. At the same time, *StreamFed-LDA* also reduces the latency by orders of magnitudes in real-world datasets.

CCS Concepts: • **Information systems** → *Data stream mining*; • **Computing methodologies** → *Distributed algorithms*; *Latent Dirichlet allocation*; • **Mathematics of computing** → *Variational methods;*

Additional Key Words and Phrases: Distributed streams, learning system, variational inference

ACM Reference format:
Yunyan Guo and Jianzhong Li. 2021. Distributed Latent Dirichlet Allocation on Streams. *ACM Trans. Knowl. Discov. Data.* 16, 1, Article 9 (June 2021), 20 pages.
https://doi.org/10.1145/3451528

1 INTRODUCTION

Topic Modeling is a popular unsupervised **machine learning** (**ML**) algorithm that can discover latent topics on collections of observed documents. **Latent Dirichlet Allocation** (**LDA**) [9], as

This work is funded by the National Natural Science Foundation of China (NSFC) Grant Nos 61832003 and U1811461.
Authors' addresses: Y. Guo, Harbin Institute of Technology, 92 Xidazhi St, Harbin, Heilongjiang 15001, China; email: guoyunyan@stu.hit.edu.cn; J. Li, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong 518055, China; email: lijzh@hit.edu.cn.

1556-4681/2021/06-ART9 $15.00
https://doi.org/10.1145/3451528

1995	2000	2005	2010	2015	2017
communicated	svm	svm	svm	lasso	mnist
fast	svms	svms	lasso	svm	lasso
unpredictable	mnist	lasso	svms	mnist	svm
encoders	kkd	minist	mnist	**admm**	**admm**
option	covariation	libsvm	spam	multilabel	gpu
relate	hyperspectral	kpca	lmnn	nesterov	github
covert	lvm	multilabel	nesterov	svms	nesterov
advantageous	leveraged	kkt	tsvm	gpu	rmsprop
remained	eigenproblem	semisupervised	kpca	rcv	nnz
ergodic	gmm	rcv	libsvm	nnz	gpus
broadens	laplacians	tsvm	candes	sbm	dnns
feeding	lasso	ohsumed	norb	**mapreduce**	sbm
earth	gmms	nystrom	biomarkers	github	python
specializization	halfspace	silp	seismic	rmsprop	svms
throughput	spam	gmms	rcv	rakhlin	**federated**
intuition	cpe	lmnn	poker	biclustering	rcv
placement	bioinformatics	viral	semisupervised	python	**hogwild**
adapter	minibatch	downside	kkt	candes	**straggler**
offering	analyticity	norb	ohsumed	negahban	candes
slice	logitboost	spam	silp	lmnn	rakhlin

Fig. 1. An example of the evolving topic on NeurIPS corpus. We learned topics with articles published from 1990 to 2017 and listed the top 20 important words for the same topic discovered by *StreamFed-LDA*.

the most important topic modeling formulation, shows its empirical power in the previous years. It assumes that each *latent topic* corresponds to a mixture assignment of words, and each document corresponds to a mixture assignment of topics. LDA has been widely used in different fields of ML applications, such as text summarization, and event detection and tracking [4, 15].

Despite the success of LDA and its variants have achieved in these applications, several important challenges need to be addressed for LDA to work with *massive streaming* datasets in real-world deployments. In this article, we investigate the following three primary challenges: *real-time inference*, *topic evolution*, and *data turbulence*.

Challenge 1: Real-time Inference. LDA has been considered to be time-consuming, especially when dealing with massive data, where it becomes highly expensive and sometimes unacceptable to access or store the full historical data in real-time. Real-world applications typically require *real-time inference* of topic modeling on streaming data, the results of which will be transferred to downstream tasks promptly. Therefore, it is highly desirable to develop models that are of *low-latency*, and at the same time maintain *high-quality* inference results on most recent data instances.

Challenge 2: Topic Evolution. Many concepts in the real-world are evolving over time, which is often reflected as *evolving topics* on streaming data such as news, social media, and scientific literature [8, 14]. To demonstrate this phenomenon, we took NeurIPS conference articles published from 1990 to 2017,[1] and inspected the shifting of word distribution over time for a specific topic. Figure 1 shows the top 20 words associated with the same topic in different years. As can be seen, the words concentrate on non-distributed solutions (e.g., "gmm", "kpca") before 2010. After 2015, parallel and distributed algorithm design became a popular research direction of this topic (e.g., "admm", "mapreduce"). If one uses the topics learned in 2010 to infer an article published in 2017, the inference quality is unsatisfied.

Challenge 3: Data Turbulence. Real-world events, especially the breaking ones, often have a big impact on the topic distribution of data, resulting in *data turbulence* on streams. Taking news articles as an example shown in Figure 2, "Iran attack", as one of the topics, attracts attention

[1] https://www.kaggle.com/benhamner/nips-papers/data. The details of the experimental settings are given in Section 6.3.

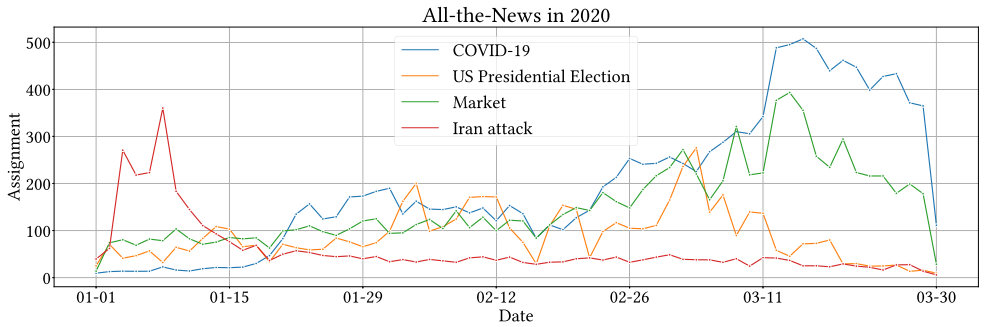

Fig. 2. An example of data turbulence. We infer the news published in the 1st season in 2020 and summarize the topic assignments of the news articles by date. The topic distributions are unstable/fluctuating across different weeks.

when related events happen, while information about "COVID-19" and "Market" explodes from late February to March with the spread of coronavirus in the United States. That is, given time duration, the topic distribution depends on the average topics' assignments of data instances. Because of these real-world events, the topic distribution in a short duration (e.g., one week) often has a significant divergence from the topic distribution over the whole stream (e.g., one year).

Data turbulence brings a new challenge to lifelong learning systems on streams. If the implicit semantics of the topics are remixed or exchanged in a short time, the inferred topic distribution of the same instance could not be *consistent*, resulting in poor *topic consistency*, which is a major criterion of evaluating topic models on turbulent streams.

LDA on Streams. ***Streaming Variational Bayes*** **(SVB)** [11] represents the state-of-the-art methods to learn latent variables on streams. It introduced *Bayesian updating* to avoid accessing massive historical data in each *round*: given the trained model as a prior and the new data instances (with the same timestamp) as observations, SVB calculates the approximate posterior as an updated model. However, *SVB-LDA* assumes that topics and the data distribution are fixed, which means it did not consider the topic consistency on turbulent streams. When the topic distribution on new data instances significantly differs from the topic distribution on data instances with the last timestamp, the learned prior is close to the random initialization. In that case, *SVB-LDA* has to learn the new model with a fuzzy topics sketch and a batch of new data instances. Therefore, it is incapable to maintain the topic consistency, and the latency of *SVB-LDA* is unacceptably high.

Contributions. To that end, we propose *StreamFed-LDA*, a distributed LDA algorithm on streams. *StreamFed-LDA* addresses above real-world challenges spontaneously for massive streaming datasets.

Following SVB, *StreamFed-LDA* uses the Bayesian updating to train the streaming datasets. To obtain high-quality inference results on turbulent streams, *StreamFed-LDA* introduces *stochastic optimization*, which ensures to preserve the learned knowledge from historical training and update the model with each new data instance. In this way, *StreamFed-LDA* attracts the new information from the latest data instances and maintains topic consistency for LDA on turbulent streams.

Moreover, the latency of *StreamFed-LDA* should be substantially optimized for real-time inference. Therefore, we further introduce the following techniques into *StreamFed-LDA* to reduce its latency. Firstly, the training process of *StreamFed-LDA* is time-consuming. The bottleneck comes from the expensive communication cost in distributed systems. To reduce the communication cost, *StreamFed-LDA* introduces *federated learning* to shrink the number of communication rounds and

modified *AllReduce* to decrease the cost of each communication round. Secondly, compared with SVB-LDA, *StreamFed-LDA* allows each machine to update the topic model frequently. The computation cost of each machine is determined by the cost of each update step and the update step size. In order to further increase the *throughput* of *StreamFed-LDA*, *StreamFed-LDA* speedups the computation for each data instance with *sparse updating* and employs an *adaptive strategy* for determining the proper update step size adaptively.

StreamFed-LDA follows the dataflow and interface design of the modern streaming platforms, in order to make it convenient to integrate the proposed algorithm into the mature streaming systems in the future [12, 40]. In summary, the main contributions of this article are as follows:

(1) We point out three primary challenges for LDA on streaming data, which are of particular importance in real-world deployment. This will hopefully attract much more attention and stimulate future work in this area.
(2) To address these challenges, we introduce *StreamFed-LDA*, a novel distributed LDA algorithm on streams. *StreamFed-LDA* learns the evolving topics on turbulent streams effectively to provide high-quality inference results. Moreover, *StreamFed-LDA* is efficient for real-time inference on massive streaming-fashion application scenarios.
(3) Experiments show that the inference quality and the topic consistency of *StreamFed-LDA* are significantly better than the baselines, and the latency is orders of magnitude lower.

The rest of this article is organized as follows. Section 2 presents the existing works. Section 3 prepares background knowledge. Section 4 explains the proposed distributed stream learning framework and introduces a novel distributed LDA algorithm on streams, named *StreamFed-LDA*. Section 5 further reduces the latency of *StreamFed-LDA* in different aspects. The experimental evaluations are presented in Section 6. Section 7 concludes this article.

2 RELATED WORKS

To handle the real-time inference on massive streams, some distributed stream processing systems separate the workload into two stages: the *training stage* and the *inference stage*. In this way, the training stage acts as a pre-processing step. The *latency* will only include the time of inference, which can be further reduced with distributed computation. One major limitation of this framework is that the inference results fully depend on topics extracted from historical data, while ignoring the evolution of topics in streams. To obtain *high-quality* real-time inference, topic evolution implies the need for lifelong learning (continual learning) on streams. Therefore, existing LDA algorithms in distributed stream processing systems, like *Flink-LDA*[2], become sub-optimal.

On the other hand, prior works, like *dynamic topic modelings*, have investigated the discovery of evolving topics on small and historical datasets with timestamps [2, 18, 22]. Compared with classic LDA, they modified the dependent relationship between the latent variables in order to design new topic modeling formulations, and following the derivation of the classic LDA algorithm, they derived similar update functions to learn the new model. These algorithms, however, cannot be readily deployed in real-time scenarios with streaming data, because the training steps that process the whole datasets are time-consuming [2, 22], and the requirement of the storage is expensive when data increases [18].

For turbulent streams, the topic distribution on new data instances is not identical to that on the sampled data instances from the whole stream. In that case, ***population variational Bayes*** (***PP-LDA***) [24] and ***trust-region*** (***TR-LDA***) [33] are proposed for LDA on streams to handle data turbulence. *PP-LDA* defined the population posterior for streams, but it stores the proportion of

[2]https://flink.apache.org/, https://github.com/alibaba/Alink.

Table 1. The Comparison between Existing Works and Proposed *StreamFed-LDA*

	training on streams	real-time inference	topic evolution	data turbulence
Flink-LDA		✓		
Dynamic TMs			✓	
PP/TR-LDA				✓
SVB-LDA	✓			
StreamFed-LDA	✓	✓	✓	✓

all seen data, which is a large-scale record with millions of features in practice. *TR-LDA* uses a trust-region update, which requires iterative sample data instances from the whole historical data uniformly, so the sampling costs increase with growing streams. Although *PP-LDA* and *TR-LDA* have the theoretical guarantee, they both require expensive storage space for massive historical records and the heavy I/O costs for sampling in each round. Therefore, none of them is practical for massive streaming of datasets.

SVB-LDA has no storage requirements or heavy I/O costs, compared with *PP-LDA* and *TR-LDA*. It is a truly streaming procedure, but the performance is unsatisfied with three other real-world challenges. We compare *StreamFed-LDA* with existing works given in Table 1 and Section 6.1.

3 BACKGROUND

Latent Dirichlet Allocation. LDA is a probabilistic generative model of documents. Given an input corpus D and the number of topics K, LDA aims at representing each document d as a mixture over topics, and model each topic as a distribution over the vocabulary V. Also, LDA infers the topic assignment of each word in each document.

LDA assumes the generative process of topics and documents. β is word proportions of K topics, a $K \times V$ matrix. kth row represents topic k. $\beta_k \sim Dirichlet(\eta)$. Each β_k is drawn from Dirichlet distribution with symmetric parameter η. θ is topic proportions of documents, a $D \times K$ matrix. The dth row θ_d represents topics proportions of document d. $\theta_d \sim Dirichlet(\alpha)$. Each θ_d is drawn from Dirichlet distribution with symmetric parameter α. When generating the nth word in document d, it draws a topic assignment $z_{dn} \sim Multinomial(\theta_d)$. Then, we draw a word from the word proportions of the topic z_{dn}. $w_{dn} \sim Multinomial(\beta_{z_{dn}})$. The whole corpus could be generated after repeating. From generative process, latent variables have complicated dependencies. The exact inference is intractable with massive corpus.

Variational Inference for LDA. Solving LDA problem means examining $p(\beta, \theta, z|w)$, the posterior distribution of β, θ, and z conditioned on given documents. While the exact inference of posterior distribution is intractable, Blei et al. [9] proposed to use ***Variational Inference*** (**VI**) to find an approximate posterior distribution. VI introduces a simple and fully factorized distribution $q(\beta, \theta, z|\lambda, \gamma, \phi)$. They are new representations of exact posterior distribution, with additional variational parameters λ, γ, ϕ. The new representations could replace complicated dependencies between variables β, θ, z.

$q(\beta_k|\lambda_k)$ is a *topic-level* variational distribution for topic k. $q(\beta_k|\lambda_k) \sim Dirichlet(\lambda_k)$. Instead of inferring $p(\beta_k|w, \eta)$ directly, VI assigns a distribution family $q(\beta_k|\lambda_k)$ as an approximation. *Document-level* variational parameters are γ_d and ϕ_d. For a document d in the corpus, $q(\theta_d|\gamma_d) \sim Dirichlet(\gamma_d)$. For the nth word in d, $q(z_{dn}|\phi_{dn}) \sim Multinomial(\phi_{dn})$, where ϕ_{dn} is the topic proportions of word w_{dn}. VI aims at minimizing the distance between the real distribution and approximate distribution in KL-divergence, which equals to maximizing the ***Evidence***

Algorithm 1: VI-LDA (Batch D, λ, K, α, η)

while λ *is not converged* **do**
  for $k = 1$ **to** K **do**
    $\mathbf{E}_q[\log \beta_k] \leftarrow DirichletExp(\lambda_k)$.
  for d *in* D **do**
    $(\gamma_d, \phi_d) \leftarrow DocInfer(d, \mathbf{E}_q[\log \beta], \eta, K)$;
    for $k = 1$ **to** K, $w = 1$ **to** V **do**
      $\tilde{\lambda}^d_{kw} = n_{dw}\phi_{dwk}$
  Update $\lambda \leftarrow \eta + \sum_{d \in D_S} \tilde{\lambda}^d$
return λ.

Algorithm 2: SVB-LDA (Stream S, K, α, η)

Initialise λ^0 randomly and $t = 1$;
while *Stream S generates data as snapshot t* **do**
  Collect data instances in snapshot t as batch D^t;
  $\lambda^t \leftarrow$ *VI-LDA*$(D^t, \lambda^{t-1}, K, \alpha, \eta)$
return λ^t.

Function *DirichletExp*(λ_k):
  for $w = 1$ **to** V **do**
    $\mathbf{E}_q[\log \beta_{kw}] = \Psi(\lambda_{kw}) - \Psi(\sum_{i=1}^{V} \lambda_{ki})$
  return $\mathbf{E}_q[\log \beta_k]$.

Function *DocInfer*(d, $\mathbf{E}_q[\log \beta]$, K, η):
  Initialize γ_d randomly;
  while γ_d *not converged* **do**
    for $k = 1$ **to** K **do**
      $\mathbf{E}_q[\log \theta_{dk}] = \Psi(\gamma_{dk}) - \Psi(\sum_{k=i}^{K} \gamma_{di})$.
    for *word* w *appears in* d **do**
      for $k = 1$ **to** K **do**
        $\phi_{dwk} \propto \exp\{\mathbf{E}_q[\log \beta_{kw}] + \mathbf{E}_q[\log \theta_{dk}]\}$.
      Normalize for w with condition $\sum_{k=1}^{K} \phi_{dwk} = 1$.
    for $k = 1$ **to** K **do**
      $\gamma_{dk} = \alpha + \sum_{w=1}^{V} n_{dw}\phi_{dwk}$
  return (γ_d, ϕ_d).

Lower Bound (ELBO) [16]. ELBO could be expressed by three variational parameters, λ, γ, and ϕ. Optimizing ELBO means choosing optimal values for λ, γ, and ϕ.

$$ELBO = \sum_d \{\mathbf{E}_q[\log p(\beta, \theta_d, z_d, w_d | \alpha, \eta)] - \mathbf{E}_q[\log q(\beta, \theta_d, z_d | \lambda, \gamma_d, \phi_d)]\}.$$

Algorithm 1 shows the basic method to solve LDA with VI, named *VI-LDA*[3]. With given static corpus D, λ converges after hundreds of iterations [7, 29, 39]. In each iteration, *VI-LDA* processes *DocInfer* function for every document.[4] In the beginning iterations, λ includes seldom information. Therefore, the results of the *DocInfer* are unacceptable and the average running time of this function is expensive – it hardly converges. The stochastic version of *VI-LDA* is *SVI-LDA* [16, 17]. It samples a mini-batch in D to run *DocInfer*. The benefit of *SVI-LDA* comes from updating λ frequently, compared with *VI-LDA*. However, it will not fit evolving streams since it assumes that mini-batches are sampled independently and identically from static datasets.

Streaming Variational Bayes. SVB is the state-of-the-art algorithm for VI on streams [11]. SVB uses *Bayesian updating* method to avoid sampling from massive historical datasets. Given a prior and new data instances, SVB calculates the approximate posterior with VI.

Algorithm 2 shows the details of *SVB-LDA*. *Snapshots* are data subsets that instances are grouped by timestamp. As a streaming procedure, *SVB-LDA* processes snapshots sequentially. In iteration t, λ^{t-1} is the parameter of the prior distribution of the model, which is the initial value of λ in *VI-LDA*. *VI-LDA* becomes the core step to optimize λ. After converging on snapshot t, *VI-LDA* returns the final λ as λ^t, which is the parameter of the approximate posterior distribution of the model.

4 STREAMING FEDERATED LDA

4.1 Framework and Dataflow

In general, distributed systems include a *master* node and multiple *worker* nodes. The master divides the tasks into stages, and it partitions the data instances and sends them to workers. Workers execute most of the computation jobs in parallel. When implementing LDA algorithms on the distributed streaming systems like *Flink*, the dataflow [12] is shown at the top of Figure 3. Flink

[3]Although MCMC and VI can asymptotically achieve similar losses in theory, empirically the model quality of MCMC is sometimes unsatisfactory [23, 41].
[4]The reference about Ψ function could be found here: https://en.wikipedia.org/wiki/Digamma_function.

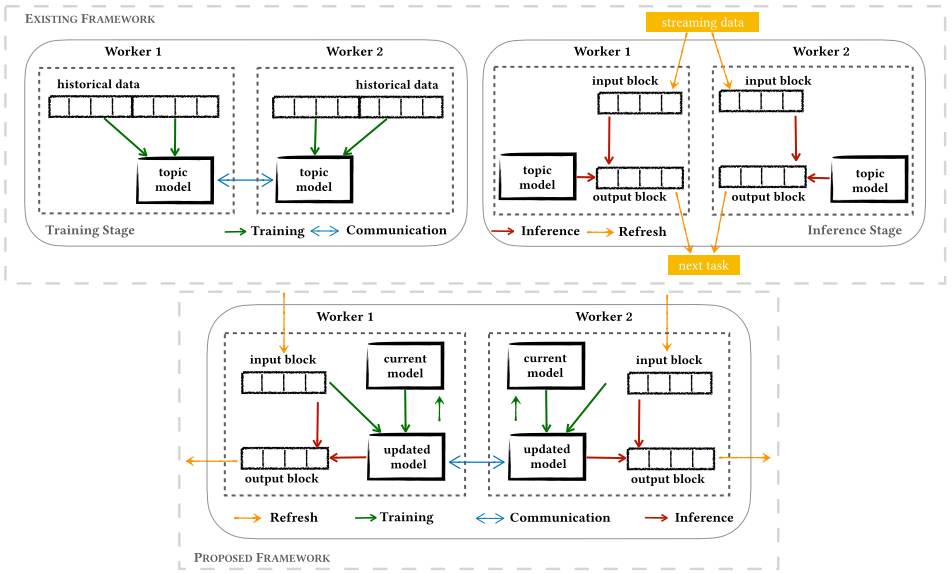

Fig. 3. The top figure shows the framework for learning and inference on distributed systems like Flink. The bottom figure represents the proposed distributed streaming framework for LDA. They include two worker nodes as an example, and the master node is omitted. The dataflows are in one round.

includes a *master* node for coordination and multiple *worker* nodes for computation. At the training stage, Flink trains topic models on static historical datasets in parallel, then each worker node loads the trained model locally. At the inference stage, the streaming data instances are partitioned and distributed into workers, and workers infer the received instances with the fixed model.

The bottom framework shown in Figure 3 presents the logic structure and dataflow of the distributed streaming framework we designed for *StreamFed-LDA* as shown in Algorithm 3. As an example, it includes two workers and omits the master node. Workers allocate four fixed-size memory spaces as the input block, the output block, the current model, and the updated model. The main stages include *refresh*, *training*, *communication*, and *inference*. The four dataflows shown in Figure 3 complete the following four stages, respectively:

(1) At the refresh stage, master partitions the snapshot and sends them to all workers, and workers store the received data into the local input blocks.
(2) At the training stage, the current model copies the updated model as the latest model. The input block data and the current model are used to calculate the updated model.
(3) Before communication, each worker holds the local updated model. In the synchronization step, the updated model of each worker is aggregated together to be the new global model. Then the global model is broadcast to all workers.
(4) When inferring, each worker uses the synchronized model to infer each data in the input block, and the corresponding output vectors are stored in the local output block.
(5) After all the data in the input blocks are inferred, the results in the output blocks are passed back to the clients or the next tasks in the application pipeline. Then, workers clear the input blocks and the output blocks to prepare for the next round.

This framework has its advantages that it allows the LDA algorithms to train the evolving topics on streaming-fashion datasets. It *merges* the training stage and the inference stage into one task in

Algorithm 3: StreamFed-LDA (Stream D, α, η, K, m)

Master:
Partition K into m sets, S_1,...,S_m;
Initialise λ^0 randomly and $t = 1$;
Broadcast λ^0 to all workers;
while *data* D^t *generated as snapshot* t **do**
  Issue *Refresh(t)* to all workers;
  Issue *TrainingTask(t)* to all workers;
  Issue *AllReduce(t)* to all workers;
  Issue *InferenceTask*(t) to all workers.

Function *LocalUpdate(D, λ, K, α, η):*
  for d *in* D **do**
    $\mathbf{E}_q[\log \beta] \leftarrow$ *PartDirichletExp*(λ, d);
    $(\gamma_d, \phi_d) \leftarrow$ *DocInfer*(d, $\mathbf{E}_q[\log \beta]$, η, K);
    Update $\lambda \leftarrow (1-\rho)\lambda + \rho(\eta + \frac{1}{|D|}(n_d \phi_d))$.
  return λ.

Function *PartDirichletExp(λ, d):*
  Initialize $\mathbf{E}_q[\log \beta]$ as zero;
  // $\mathbf{E}_q[\log \beta]$ is a $K \times V$ matrix;
  for $k = 1$ **to** K, w *appears in* d **do**
    $\mathbf{E}_q[\log \beta_{kw}] = \Psi(\lambda_{kw}) - \Psi(\sum_{i=1}^{V} \lambda_{ki})$
  return $\mathbf{E}_q[\log \beta]$.

Worker r = 1, ..., m:

Function *Refresh(t):*
  Receive a partition $D^{t,r}$ from the stream D.

Function *TrainingTask(t):*
  Receive λ^{t-1} parts from other workers;
  // from *AllReduce* in the last round;
  $\lambda^{t,r} \leftarrow$ *LocalUpdate*($D^{t,r}$, λ^{t-1}, K, α, η).

Function *AllReduce(t):*
  for $k = 1$ **to** K **do**
    Send $\lambda_k^{t,r}$ to Worker v, where $k \in S_v$;
  foreach k *in* S_r **do**
    for $v = 1$ **to** m **do**
      Receive $\lambda_k^{t,v}$ from Worker v;
    Aggregate $\lambda_k^t \leftarrow \frac{1}{m} \cdot \sum_{v=1}^{m} \lambda_k^{t,v}$;
    $\mathbf{E}_q[\log \beta_k^t] \leftarrow$ *DirichletExp*(λ_k^t);
    Broadcast $(\lambda_k^t, \mathbf{E}_q[\log \beta_k^t])$ to other workers.

Function *InferenceTask(t):*
  Receive $\mathbf{E}_q[\log \beta^t]$ parts from other workers;
  foreach d *in* $D^{t,r}$ **do**
    Output $(\gamma_d, \phi_d) \leftarrow$ *DocInfer*(d, $\mathbf{E}_q[\log \beta^t]$, η, K).

distributed stream processing systems. Therefore, different from other frameworks such as shown in Figure 3, the model learns the new information from the most recent data, and it infers the most recent data instances based on the new model.

Besides, this framework satisfies the requirement of cheap I/O cost, when algorithms do not need to access any historical data. In streaming procedures, the training stage uses the most recent data instances. In that case, we could design the distributed version of streaming LDA algorithms on this framework. We take *SVB-LDA* as an example: First, the training stage uses *VI-LDA* function; and, second, at the communication stage, the updated models are summarized to be the new synchronized model for the inference stage.[5]

4.2 Training on Streams

When designing the algorithm in the proposed framework, the training stage plays an important role since it impacts the training latency and the quality of the topic model [1, 13, 28].

However, distributed *SVB-LDA*, which uses *VI-LDA* in the training stage, is not suitable for turbulent streams. In the originally applicable scope of *SVB-LDA*, each data point is sampled independently and identically from the data stream. And in each round, the amount of the most recent data must be sufficiently large, so that the most recent data could represent the whole stream. On turbulent streams, the divergence between the most recent data and the last round data is usually huge as shown in Figure 2. Thus, if the initial model of *VI-LDA* is learned from the data in the last round, the model maintains seldom topics information of the data in this round, which is close to

[5]Notice that the original design of *SVB-LDA* did not consider the inference and communication stage on distributed stream processing systems. For fair comparison, we implement the distributed *SVB-LDA* as the same as *StreamFed-LDA* except for the training stage.

the random initialization. Even worse, the new information replaces the last model completely in the update step. Therefore, *SVB-LDA* hardly remains the topic consistency on turbulent streams.

Moreover, the training latency of *SVB-LDA* becomes high because of the almost random initialization. The inner iterations of *VI-LDA* stop when the model λ converges. *Inner iteration* includes two steps: using the input data and the current model to calculate the intermediate results, then using the intermediate results to update the current model to generate the updated model. The optimization process could have multiple inner iterations. If it starts from the almost random initialization, the number of the inner iterations is big. Also, this number is hard to estimate and is unbalanced among all workers. In each inner iteration, *SVB-LDA* processes all data in the input blocks. Multiple reading and updating the whole model is time-consuming.

To that end, we design *StreamFed-LDA*, which uses *LocalUpdate* function at the training stage, to address the challenge of data turbulence.

The design of the *TrainingTask* function in Algorithm 3 follows the *Bayesian updating* method. The model from the last round is used as the initial value, corresponding to the prior distribution of the model. Given the most recent data instances as the observations in each round, the goal of training is to optimize the posterior distribution of the model. At the communication stage of *StreamFed-LDA*, the updated model is synchronized among all workers, which becomes the prior distribution of the model in the next round.

In the *TrainingTask*, each worker runs the *LocalUpdate* function as the optimization method to update the local model at the training stage, which is shown in function in Algorithm 3. Each instance is a sample of the data in the same snapshot. And each worker calculates the intermediate result based on each instance. The intermediate result is added to the model with a certain proportion. Thus, the *stochastic optimization* strategy is used to update the local model in each inner iteration. The *local update step size*, that is, the number of inner iterations in *LocalUpdate*, is equal to the data size in the input block.[6]

LocalUpdate brings the following advantages into *StreamFed-LDA* on turbulent streams: Using the most recent data to update the model for multiple times, the updated model deeply captures the trend of the data stream to provide high-quality inference results. The Bayesian updating and stochastic optimization together remain the trained information in history, which guarantees the topic consistency on turbulent streams.

4.3 Online Inference

In the *InferenceTask* function, each worker holds the synchronized model. The *DocInfer* function infers the data in the input block and returns (γ, ϕ) as the inference results. The topic assignments γ are stored in the output blocks. The calculation of $\mathbf{E}_q[\log \beta]$ is supposed to be part of the *InferenceTask* function. To parallel this expensive and bottleneck computation, Algorithm 3 assigns it into the modified *AllReduce* function. The benefit comes from amortizing the computation cost of the *DirichletExp* function into multiple workers.

5 LATENCY REDUCTION

Considering the efficiency of LDA, the training stage is more time-consuming than the inference stage. As a mature platform, Flink only processes the inference stage on streams, in order to satisfy the requirement of the real-time inference. On the contrary, *StreamFed-LDA* combines the training stage and the inference stage into one task, which requires more effort to reduce the training latency. Ideally speaking, if the training latency is in the same order as the inference latency, *StreamFed-LDA* will be practical for real-time inference in various applications.

[6]We fix the learning rate ρ as $\frac{1}{|D|}$ to avoid tuning.

5.1 Complexity Analysis with Sparse Updating

We split the computation in *LocalUpdate* into two parts. One is the *DocInfer* function for each document. The other is the "update" step for each document, which includes the calculation of $\mathbf{E}_q[\log \beta]$ and the update of λ.

The computation in the "update" step is dense, corresponding to the size of vocabulary: the update of λ and the calculation of $\mathbf{E}_q[\log \beta]$ are $O(K \cdot V)$, where K is user-defined topic size and V is the vocabulary size. In detail, calculating $\mathbf{E}_q[\log \beta]$ needs to read the whole matrix to get the sum of each row, and updating λ needs to write every element in it because of the decay strategy. Those two steps are expensive. However, word vectors in most documents are extremely sparse compared with vocabulary. For example, the vocabulary size V is approximately 100 K, and the average length of documents $\bar{N}$ is 300 in `NYTimes` and 97 in `PubMed` (Table 2). Thus, $\mathbf{E}_q[\log \beta]$ and (γ_d, ϕ_d), the input and output parts of *DocInfer* in Algorithm 3, are sparse matrices.

Based on the above insights, we improve the "update" step in Algorithm 3 to guarantee the read and write of λ are sparse options, then the computation cost will be reduced from $O(K \cdot V)$ to $O(K \cdot \bar{N})$. *Sparse update* strategy could ensure that the computation of *LocalUpdate* is sparse. The main idea of the sparse update strategy is to write a sparse update into a proxy matrix. And when the algorithm needs the real matrix, it reads the proxy matrix and uses a mapping function to recover the real matrix. In round t, each worker processes S documents, $s \in \{0, \ldots, S-1\}$. For processing documents, the sequence is $\lambda^0, \lambda^1, \ldots, \lambda^S$. And, $\lambda^{s+1} = (1-\rho)\lambda^s + \rho S\tilde{\lambda}^s + \rho\eta$. The introduced serial proxy matrix are: $Q^0, Q^1, \ldots, Q^S$. $Q^0 = \lambda^0$, $Q^{s+1} = Q^s + \frac{\rho S}{(1-\rho)^{s+1}}\tilde{\lambda}^s$. The mapping function from Q to λ is: $\lambda^{s+1} = (1-\rho)^{s+1}Q^{s+1} + \rho\eta\sum_{j=0}^{s}(1-\rho)^j$. The cost of the element-wise mapping function is cheap. Similarly, we maintain $RowSum(Q)$ as the proxy vector of $RowSum(\lambda)$. The update of sums is sparse with it. $RowSum(\lambda)$ is the sum values of each row. $RowSum(\lambda^0) = RowSum(Q^0)$, and

$$RowSum(Q^{s+1}) = RowSum(Q^s) + \frac{\rho S}{(1-\rho)^{s+1}} RowSum(\tilde{\lambda}^s),$$

$$RowSum(\lambda^{s+1}) = (1-\rho)^{s+1} RowSum(Q^{s+1}) + V\rho\eta \sum_{j=0}^{s}(1-\rho)^j.$$

With the above definition of mapping functions, we could read and write λ in proxy matrix Q sparsely. When we need sparse λ^{s+1}, we could read Q^{s+1} sparsely, and use the mapping function to have sparse λ^{s+1}. Then, sparse λ^{s+1} and $RowSum(\lambda^{s+1})$ together calculates part $E_q[\log \beta]$. We could write the update of $\tilde{\lambda}^s$ into proxy matrix Q^{s+1} sparsely, and fresh $RowSum(Q^{s+1})$ with sparse $\tilde{\lambda}^s$. Meanwhile, the sparse update does not have additional memory requirements. Each worker stores the proxy matrix in memory, and the real matrix exists logically.

Now we compare the *time complexity* between *InferenceTask* and *TrainingTask*. In the *InferenceTask*, the core function *DocInfer* includes a loop. The computation cost of each loop is $o(K \cdot \bar{N})$, and the convergence rate is $O(\sqrt{\bar{N}})$ [10, 17]. Therefore, the time complexity of *InferenceTask* is $\Omega(K \cdot \bar{N} \cdot S)$. The execution of *LocalUpdate*, which is the main computation of *TrainingTask*, can be partitioned into two parts: *DocInfer* and the "update" step. First, running the *DocInfer* function for each document is equivalent to the *InferenceTask*. On the other hand, the time complexity of the "update" step is $o(K \cdot \bar{N} \cdot S)$ for S documents, which is reduced by sparse updating. In conclusion, the time complexity of *TrainingTask* is of the same order as the time complexity of *InferenceTask*.

5.2 Adaptive Strategy

In the *TrainingTask*, *StreamFed-LDA* extracts the information from each instance, which means the local update step size equals the input size in each round. On turbulent streams, the benefit of this

setting is that the model could learn the new information to infer the most recent data effectively. However, if the model could infer the most recent data well before running the *TrainingTask*, it is not necessary to train the model with every data instance. In that case, choosing a *reasonable* local update step size will further reduce the training latency.

The reasonable local update step size is the optimal parameter that *StreamFed-LDA* maximizes the objective function ELBO. Unfortunately, it is hard to calculate the reasonable local update step size based on the ELBO exactly: The ELBO of each round is changing with the most recent data; besides, the requirement of the real-time inference limits the time and space costs.

Therefore, in order to estimate the reasonable local update step size approximately, we design the adaptive strategy for *StreamFed-LDA*:

(1) Before training, it calculates the ELBO based on the most recent data and the current model.
(2) It uses the information of the last round to estimate the reasonable local update step size of the current round with the following formulations.
(3) After running the *TrainingTask* and the *InferenceTask*, the ELBO value and the local update step size value are stored to prepare for the next round.

The approximate reasonable local update step size τ_t is calculated from the one in the last round:

$$\tau_t = \begin{cases} \tau_{t-1} \cdot \sqrt{\frac{L_{max}-L(\lambda_t)}{L_{max}-L(\lambda_{t-1})}}, & \text{if } L(\lambda_{t-1}) \leq L(\lambda_t); \\ \tau_{t-1} \cdot \sqrt{\frac{L_{max}-L(\lambda_t)}{L_{max}-L(\lambda_{t-1})}} - c, & \text{others,} \end{cases}$$

where $L(\lambda_t)$ is the ELBO in round t before training, L_{max} represents the estimated upper bound of the ELBO, and c is a positive constant. It is inspired by *Adaptive Communication* [34]. According to Theorem 2 in [34], we derive the following equation: $\frac{\tau_{t_1}}{\tau_{t_2}} = \sqrt{\frac{L_{max}-L(\lambda_{t_1})}{L_{max}-L(\lambda_{t_2})}}$, t_1 and t_2 represent any two rounds. τ_{t_1} and τ_{t_2} mean the ratio of the "distance" between the objective function value and the lower bound of the objective function.

The main idea of the adaptive strategy is to determine the increase or decrease trend of the local update step size, according to the difference between the ELBO of the last round and the ELBO of the current round.

If the ELBO of the last round is lower than the ELBO of the current round, it indicates that the current model could infer the most recent data better than the model in the last round, and the local update step size can be appropriately decreased to reduce the latency and to avoid overfitting.

On the other hand, if the ELBO of the last round is larger than the ELBO of the current round, it indicates that the model infers the most recent data worse than the last round. There are two possible causes. The first one is that the most recent data are significantly different from the historical data, which makes the model unable to make good inference results for the most recent data. Therefore, the adaptive strategy increases the local update step size appropriately based on the difference between the current ELBO and the last ELBO. Secondly, although the most recent data are similar to the historical data, the latest local update step is so large that the model is divergent. Therefore, the local update step size could be linearly reduced to make the model converge.

The adaptive strategy introduces an estimation step into *StreamFed-LDA*. In the estimation step, the computation cost of the ELBO is $O(u)$, where u is the input block size in each worker. To further reduce the estimation cost, we sample 10% data from the most recent data to compute an approximate ELBO. It is effective and efficient enough for turbulent streams.

Although adaptive learning rate is also important for some cases [30], the goal of the adaptive strategy in this article is to reduce latency [42].

5.3 AllReduce Synchronization

People have been proposing various ways of running ML algorithms in a distributed manner [21, 43]. Motivated by *Federated Averaging* (FA) [25], *StreamFed-LDA* averages the local updated models from all workers at the end of the training stage. And then the averaged model is broadcast to all workers. Unlike FA, the master activates all workers that receive new streaming data partitions to run the *TrainingTask* function, not only random subsets of workers. Also, the aggregating triggers happen spontaneously at the same time among workers, because the computation cost of *LocalUpdate* is balanced in *StreamFed-LDA*.

Synchronization and *asynchronization* are two strategies when aggregating in distributed systems. With synchronization, the master is the bottleneck since all workers send huge results to the master roughly at the same time. The master computes the accurate aggregation results with that expensive cost. On the contrary, with asynchronization, the master adds partial results as soon as any worker has done the computation [36, 38, 39]. In that case, the master is not a bottleneck anymore, but the master always sends the stale models to the workers [19, 37]. In the framework we proposed, the staleness of the trained model is unacceptable for inference, since it causes the inconsistent inference results of the data instances which hold the same timestamp.

Considering both aspects, *StreamFed-LDA* implements the modified *AllReduce* [3, 6] as the aggregating function. The *AllReduce* function partitions the aggregation job by topics. Each worker aggregates part of the matrix of the model, and broadcasts the part result to other workers. First, aggregating results are always correct. As one of the synchronization implementations, the *AllReduce* strategy guarantees the consistency of the global model among workers when inferring. Secondly, it frees the master from aggregation jobs. Also, when the worker size increases, the computation cost on each worker remains the same. Thus, *AllReduce* improves the scalability of our framework. With balanced computation costs of *LocalUpdate* in Algorithm 3, the *AllReduce* function could be faster than some asynchronization situations as well.

6 EXPERIMENTS

We now evaluate *StreamFed-LDA* in our framework with the following experimental setups.

Cluster. We conduct our experiments on a cluster with 33 machines, one master and 32 workers. Each machine has one 1.90 GHz CPU, 24 GB of memory, and 1 Gbps network interface.

Datasets. We use `NeurIPS`, `All-the-News`, `New York Times` (`NYTimes`), and `Public Medicine` (`PubMed`)[7] datasets in the experiments, as shown in Table 2. All datasets are from public repositories. The former two datasets are real-world streams with timestamps, which maintain the characteristics of topic evolution and data turbulence naturally. We extend the trend of the turbulence of the latter two datasets to demonstrate the scalability.

Implementations. We implement the proposed framework with C++ and OpenMPI [32]. We implement *StreamFed-LDA*, *SVB-LDA*, and *TR-LDA* into the proposed framework. *SVB-LDA* and *TR-LDA* follow the original open-source project.[8] *SVI-LDA* and *Flink-LDA* follow the *Online-LDA* in *Spark MLlib*[9] [26]. The same data structures and math libraries guarantee comparison fairness.

6.1 Performance

Topic Consistency Comparison. We first evaluate the topic consistency of *StreamFed-LDA* and *SVB-LDA* on turbulent streams. The test datasets are sampled from the streams uniformly. The

[7] https://www.kaggle.com/benhamner/nips-papers/data, https://components.one/datasets/all-the-news-2-news-articles-dataset/, https://archive.ics.uci.edu/ml/machine-learning-databases/bag-of-words/.
[8] https://github.com/tbroderick/streaming_vb, https://github.com/lucastheis/trlda.
[9] https://github.com/apache/spark.

Table 2. Statistics of Datasets

Dataset name	corpus size	# documents	# tokens	avg doc length	vocabulary size
NeurIPS	0.2 GB	7.2 K	4.7×10^6	643	13.2 K
All-the-news	9.2 GB	2.6 M	7.5×10^8	279	56.3 K
NYTimes	9.6 GB	3.0 M	1.0×10^9	300	102.7 K
PubMed	7.3 GB	8.2 M	7.4×10^8	97	141.0 K

same test dataset is used to measure *loglikelihood* on the evolving model. The value of *loglikelihood* indicates the quality of the model.

To evaluate the model quality of *StreamFed-LDA*, we provide the ideal log-likelihood value that learned from *SVI-LDA*. *SVI-LDA* is the state-of-the-art LDA algorithm on static datasets. The streaming data are gathered to be static datasets. We tuned the learning-rates of *SVI-LDA* multiple times to select the optimal log-likelihood value.

From Figure 4, the *loglikelihood* values are converged in *StreamFed-LDA*, but they are not converged in *SVB-LDA*. The model of *StreamFed-LDA* converges to a practical solution that is almost acceptable with several rounds. After that, despite the turbulence of the topic distribution on the streams, the *loglikelihood* values on the test datasets are stable near the ideal values. The *loglikelihood* values of *SVB-LDA* in each round changes drastically with the turbulence of data distribution, showing a phenomenon of divergence. The curve of *SVB-LDA* is much lower than *StreamFed-LDA*.

In conclusion, the lines of *StreamFed-LDA* show a convergence trend, indicating that the models have strong consistency over time. On the contrary, the models of *SVB-LDA* on the turbulent streams are not consistent. For the turbulent streams, the learned models of *SVB-LDA* overfit on the most recent data. But the topics information that appeared before has been ignored, which turns to the divergent model. However, *StreamFed-LDA* combines Bayesian updating and stochastic optimization in *TrainingTask*, which better retains the former information that does not involve in the most recent data. As the intermediate results in application pipelines, the outputs of *StreamFed-LDA* are highly practical.

Inference Quality Comparison. Next, we show the online inference quality of *StreamFed-LDA*, compared with *SVB-LDA* on turbulent streams. The synchronized model infers the most recent data on workers in each round. Because the data change over time, we measure the inference quality with *perplexity*, instead of *loglikelihood*. *Perplexity* amortizes the inference quality of the model into each word. The value of *perplexity* represents the randomness of each word that could be generated from vocabulary by the trained model: $perplexity = exp\{-\frac{loglikelihood}{token}\}$. The smaller the perplexity value, the better the model is. And the inference quality on the most recent data guarantees the utility of the algorithm.

From Figure 4, the perplexities of *StreamFed-LDA* are better than those of *SVB-LDA* in almost all rounds. In *StreamFed-LDA*, the model needs to train several iterations to achieve a similar perplexity value of *SVB-LDA*. After that, the results of *StreamFed-LDA* is better than *SVB-LDA* when the data is turbulent.

We conclude that the inference quality of *StreamFed-LDA* is better than *SVB-LDA*. *StreamFed-LDA* and *SVB-LDA* update the model from the most recent data multiple times, so the inference qualities are generally satisfied. The slight advantages come from the limited local update step size in *StreamFed-LDA*. *SVB-LDA* requires each worker to return the local optimal model, but the optimal models are probably not the same one, so the quality of the synchronized model could

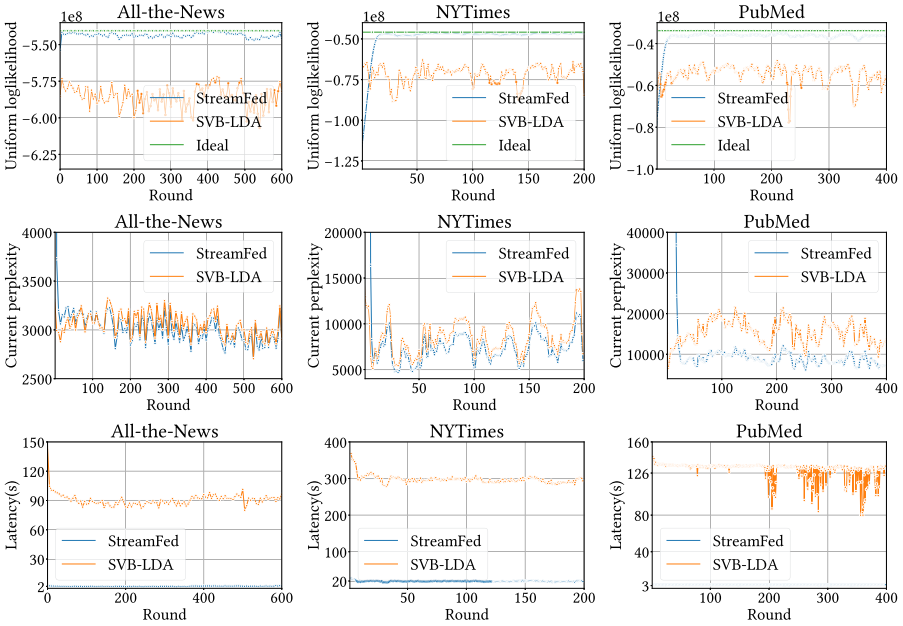

Fig. 4. Comparison between *SVB-LDA* and *StreamFed-LDA*.

be worse than the optimal. Therefore, *StreamFed-LDA* is more suitable for distributed streaming learning and online inference.

Latency Comparison. Then, we compare the latency of *StreamFed-LDA* and *SVB-LDA*. The latency includes the running time of the training stage and the communication stage. Except for the *TrainingTask*, the running time of the others are the same between *StreamFed-LDA* and *SVB-LDA*.The maximum size of the inner iterations is set to 40 in *SVB-LDA*. In Figure 4, the average latency of *SVB-LDA* is about 50, 15, and 40 times longer than that of *StreamFed-LDA* over three datasets. In *SVB-LDA*, the numbers of the inner iterations with `All-the-News` and `NYTimes` reach 40 in almost all rounds, and it decreases but is still above 20 in the later rounds in `PubMed`. The latency of *StreamFed-LDA* is stable, compared with *SVB-LDA*.

In conclusion, the latency of *StreamFed-LDA* is an order of magnitude lower, more stable, and easier to estimate than *SVB-LDA*. It is because that *SVB-LDA* needs to run multiple inner iterations until converges. In each inner iteration, the cost of calculating the intermediate result is $o(S \cdot \bar{N} \cdot K)$ and the cost of updating the model is $o(V \cdot K)$. In *StreamFed-LDA*, however, stochastic optimization and sparse computing both reduce cost to $o(S \cdot \bar{N} \cdot K)$. Therefore, the *TrainingTask* running time in *StreamFed-LDA* is approximately the same as one inner iteration cost in *SVB-LDA*. Since the model size of `NYTimes` is bigger than that of `PubMed` and `All-the-News`, the benefits of *StreamFed-LDA* with `NYTimes` is decreased because of the communication cost.

Comparison with Other Baselines. We further compared *StreamFed-LDA* with two other baselines we mentioned in the related works. Algorithms access the data points in `All-the-News` by date to simulate the real-world streaming. Figure 5 shows the comparison results.

Flink-LDA trained the model on the historical data which includes the data in the first 25 rounds and used the fixed model to inference the whole streaming data. The latency of *Flink-LDA* only depends on the inference process, while the latency of *StreamFed-LDA* includes both training and

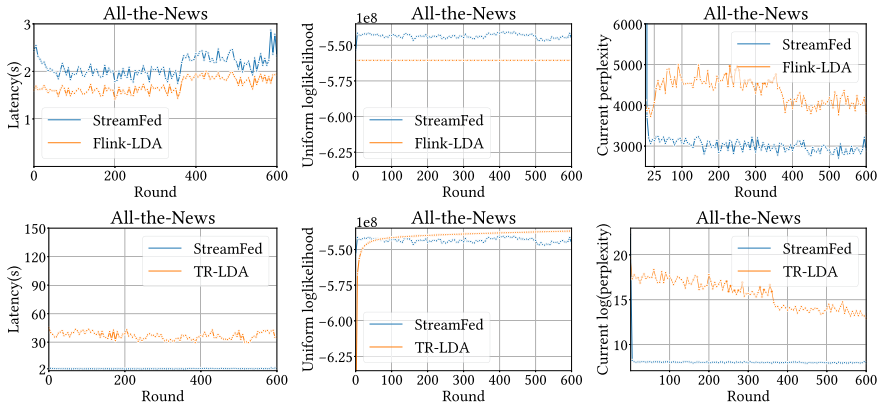

Fig. 5. Comparison between *Flink-LDA*, *TR-LDA* and *StreamFed-LDA*

inference. The comparison implies that the computation complexity of *StreamFed-LDA* is identical to the computation complexity of the inference function. Therefore, even though *StreamFed-LDA* includes the online training process, it satisfies the requirement of real-time inference. Since the model of *Flink-LDA* only obtains the historical information, the performance on both test data and streaming data is worse than *StreamFed-LDA* in the right two plots shown in Figure 5.

The latency of *TR-LDA* is unacceptable since it trains the sampled data until it converged locally. The sampled data points are sampled from the whole historical dataset, which brings more storage requirements with the increasing streaming data. *TR-LDA* keeps learning from the sampled data, so its performance on the whole historical dataset is the best, which has been clarified by its original article. However, the new information about the streaming data could not be captured as soon as possible, so the performance of *TR-LDA* is the worst as shown in the last plot of Figure 5.

6.2 Scalability

Impact of the Local Update Step Size. To explore the impact of the local update step size in *StreamFed-LDA*, we fix the worker size and increase the local update step size. Table 3 presents the corresponding latencies and throughputs on average. *Throughput* means the document counts that *StreamFed-LDA* could process per second. $throughput = \frac{u \cdot m}{latency}$, where m is the worker size.

In Table 3, the latency grows with the local update step size, since the running time of *Training-Task* is proportional to the local update step size. When processing the same amount of documents, bigger local update step sizes also imply fewer number of rounds. So the throughput grows because of fewer communication costs.

Impact of the Worker Size. Generally, when the speed of data generation increases, increasing the number of workers is an option to increase throughput. We now explore the relationship between the throughputs of *StreamFed-LDA* and the worker sizes in the distributed systems.

We increase the worker size from 1 to 32 and record the average values of the latency and the throughput in Table 4. Two comparisons are *weak-scale* and *strong-scale*. The local update step size is the same in weak-scale comparison, so the size of the processing data in each round increases with the growing worker size. On the other hand, the processing data size is the same in strong-scale comparison, so the local update step size decreases with the growing worker size.

In the weak-scale comparison, the throughput grows approximately linearly, since the processing speed of each worker essentially remains the same. The slightly increasing part in the latency

Table 3. Efficiency with Local Update Step Sizes

NYTimes	Local steps	0.5k	1k	2k	4k	8k
	Latency(s)	7.9	11.8	20.0	38.4	74.8
	Throughput	504.8	677.9	794.9	828.7	852.1
PubMed	Local steps	312	625	1.25k	2.5k	5k
	Latency(s)	2.6	3.0	3.7	5.2	8.3
	Throughput	1904.5	3333.1	5334.5	7501.2	9350.9

Table 4. Scalability with Increasing Worker Sizes on PubMed

	worker size	1w	2w	4w	8w	16w	32w
weak-scale	local steps	2.5k	2.5k	2.5k	2.5k	2.5k	2.5k
	latency(s)	0.97	4.0	4.3	4.7	5.2	5.1
	throughput	2582.3	1259.4	2321.5	4289.8	7501.2	15567.9
strong-scale	local steps	40k	20k	10k	5k	2.5k	1.25k
	latency(s)	12.4	21.1	12.2	7.5	5.2	3.7
	throughput	3227.6	1893.8	3277.0	5344.6	7501.2	10794.3

comes from the communication cost, which is increased by workloads growing in the network. Therefore, the weak-scale solution is suitable for the purpose of increasing the throughput, when the speed of data generation increases.

In the strong-scale comparison, the latency decreases with the reduced local update step size. However, due to the slight increase in communication costs, the changing rate of the latency and the throughput will gradually diminish. So the strong-scale solution is suitable for reducing the latency in a certain situation.

The single-node system has no communication cost and aggregation cost. The latency only depends on the local update step size and the initialized calculation in each round. However, the throughput of *StreamFed-LDA* with a single node is limited, which is smaller than that with 4 or 6 workers in distributed systems.

Effects of Adaptive Strategy. To demonstrate the advantages of the adaptive strategy, we fix the input block size and compare the efficiency of *StreamFed-LDA* with and without the adaptive strategy. From the left plots shown in Figure 6, the adaptive strategy chooses reasonable smaller local update step sizes, though the input block sizes are large. Table 5 shows that the adaptive strategy reduces the training cost significantly. Although it introduces estimation cost, the latency is lower in total. Meanwhile, the topic consistency and inference quality basically remain the same from the right two plots shown in Figure 6. It implies that the income of the adaptive strategy does not have to sacrifice its performance.

In conclusion, the adaptive strategy is considerably efficient for decreasing the latency, when the input block sizes are large and the training costs are high.

6.3 Topic Evolution Extraction

The historical trend of the research directions in NeurIPS conference papers could be represented by the evolving topics. We use the NeurIPS corpus to demonstrate the topic evolution discovered by *StreamFed-LDA*. The topic size is 30 in the topic evolution experiment. We tuned this hyper-parameter from 10 to 200 using the log-likelihood value, and 30 is the best choice. The vocabulary

Table 5. Performance of Basic *StreamFed-LDA* (Basic) and Adaptive *StreamFed-LDA* (adap)

	NYTimes basic	NYTimes adap	PubMed basic	PubMed adap
throughput	852.1	**1149.9**	9350.9	**11757.9**
latency(s)	74.8	**55.7**	8.3	**6.8**
estimation(s)	–	3.3	–	0.4
training(s)	66.4	46.1	5.7	3.9
communication(s)	8.3	6.1	2.6	2.5

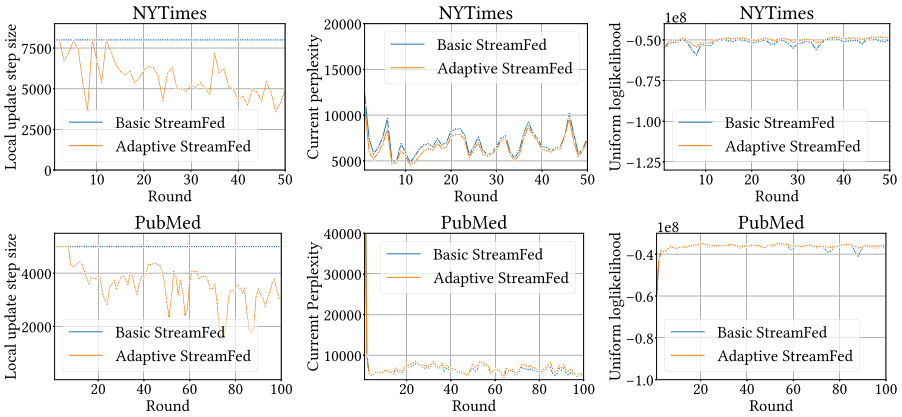

Fig. 6. Performance of the adaptive strategy on the NYTimes and PubMed datasets.

is generated from abstracts, and articles are ordered by the publishing year. The master activates two worker nodes to train and infer, and the input block size is set to 100.

Figure 1 shows the evolution of the top 20 important words for one topic among 20 years. The bolded words are the evidence that the research direction of parallel and distributed learning has become popular since 2015. To make the results more convincing, we show more topic evolution examples in appendix Figure 7. We name the discovered topics we showed as "linear models", "computer vision", and "neural networks". We conclude that the topics are evolving in real-world scenarios, and *StreamFed-LDA* can capture the trend.

For LDA, the topic size is one of the user-defined hyper-parameter. In other scenarios, the topic size could be changing over time, and the extension formula with dynamic topic sizes brings more challenges. In this article, *StreamFed-LDA* improves the performance of the LDA formula in distributed streaming settings. And it is worth studying whether the techniques we proposed could be extended to other complicated formulas to improve the performance as well. Therefore, we leave this investigation for future work.

Another relevant research area is *Event Detection and Tracking* [5, 27]. It has been wildly studied in recent years [20, 31, 35]. LDA is one of the basic document processing tasks in those application pipelines. And it is promising that *StreamFed-LDA* could improve those applications in the industry.

7 CONCLUSION

In conclusion, we proposed *StreamFed-LDA*, a novel distributed algorithm for LDA on massive streaming datasets. In real-world applications of LDA, *StreamFed-LDA* addresses three challenges

well: *StreamFed-LDA* learns the evolving topics effectively, provides high-quality inference results on turbulent streams and infers the new data instances in real-time. The experimental results on four real-world datasets show that *StreamFed-LDA* achieves significantly better inference quality compared with SVB-LDA, and the latency of *StreamFed-LDA* is orders of magnitudes lower than baseline. *StreamFed-LDA* could be effortlessly deployed in distributed environments, and the update step of *StreamFed-LDA* could be modified for various LDA. It is promising to implement *StreamFed-LDA* into the ML library of Flink, or other distributed streaming systems, to serve application pipelines widely. Also, the techniques of *StreamFed-LDA* could inspire future work on improving other ML algorithms when dealing with massive streaming data in real-world deployments.

APPENDIX

1995	2000	2005	2010	2015	2017
x_i	rbm	eigenmaps	rbm	cifar	cifar
mvl	noising	ppca	mmd	minibatch	gans
accelerates	smt	noising	eigenmaps	rbm	vae
electroencephalogram	ppca	rbm	rbms	**mmd**	**svhn**
practitioner	minibatch	mmd	ppca	rbms	mmd
behalf	denoise	smt	cifar	nade	wgan
plateau	rbms	mles	smt	vae	minibatch
organic	sbn	minibatch	mmsb	theano	gru
weinberger	denoised	retailer	minibatch	svhn	vaes
ahead	dilated	denoise	noising	blizzard	celeba
system	x_i	milder	dca	smt	pixelcnn
tendency	mvl	rbms	necessitated	**gru**	batching
necessitating	gmrf	beltrami	blizzard	batching	resnet
explicit	plenty	sbn	ggms	eigenmaps	rbm
pervasive	accelerates	denoised	convnets	noising	**openai**
resemblance	electroencephalogram	multiview	convnet	mmsb	lsun
possibly	practitioner	dilated	snes	ppca	**theano**
completion	behalf	x_i	mles	vaes	**resnets**
mmd	plateau	relatedness	hyperspheres	walkback	spn
apprenticeship	organic	mvl	ssl	advi	steganography

1995	2000	2005	2010	2015	2017
cell	cell	rmn	arow	imagenet	imagenet
rmn	rmn	cell	disaggregation	amazon	resnet
api	api	api	sentiment	sentiment	crowdsourcing
fab	fab	friendship	amazon	**crowdsourcing**	**amazon**
fdr	fdr	fab	imagenet	adagrad	nmt
network	network	fdr	adagrad	appliance	adagrade
synapsis	synapsis	network	appliance	rcnn	sentiment
cortical	cortical	synapsis	rmn	disaggregation	adapter
eye	eye	cortical	crowdsourcing	downpour	relus
single	single	eye	sustainability	relus	rcnn
field	field	single	gpus	collaboratively	appliance
activity	activity	field	lsvrc	**rmn**	sustainability
lnp	lnp	activity	isotonic	lsvrc	disagrregation
mean	mean	lnp	facebook	arow	odometer
synaptic	synaptic	mean	mcts	curation	**downpour**
signal	signal	synapsic	assortment	hmdb	rmn
closed	closed	signal	influencers	sustainability	**metagrad**
right	right	closed	pacman	freebase	**collaboratively**
theory	theory	right	dagger	yorker	lsvrc
cortex	cortex	theory	netflix	corwdclustering	hmdb

Fig. 7. Two examples of the evolving topics on NeurIPS corpus.

ACKNOWLEDGMENTS

The authors would like to thanks Dr. Jiang Guo at MIT for improving this work with his wisdom and enthusiasm.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xianqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. ACM, 265–283.

[2] Ayan Acharya, Joydeep Ghosh, and Mingyuan Zhou. 2018. A dual Markov chain topic model for dynamic environments. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1099–1108.

[3] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. 2014. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research* 15, 1 (2014), 1111–1133.

[4] Charu C. Aggarwal. 2013. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*, Charu C. Aggarwal and Chandan K. Reddy (Eds.). CRC Press, 231–258.

[5] James Allan, Ron Papka, and Victor Lavrenko. 1998. On-line new event detection and tracking. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 37–45.

[6] Michael Anderson, Shaden Smith, Narayanan Sundaram, Mihai Capotă, Zheguang Zhao, Subramanya Dulloor, Nadathur Satish, and Theodore L. Willke. 2017. Bridging the gap between HPC and big data frameworks. In *Proceedings of the VLDB Endowment*. 901–912.

[7] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. 2017. Variational inference: A review for statisticians. *The Journal of the American Statistical Association* 112, 518 (2017), 859–877.

[8] David M. Blei and John D. Lafferty. 2006. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*. 113–120.

[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *The Journal of Machine Learning Research* 3 (2003), 993–1022. http://jmlr.org/papers/v3/blei03a.html

[10] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. 2016. Optimization methods for large-scale machine learning. *SIAM Rev* 60, 2 (2018), 223–311. DOI: 10.1137/16M1080173

[11] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C. Wilson, and Michael I. Jordan. 2013. Streaming variational bayes. In *Proceedings of the Advances in Neural Information Processing Systems*. 1727–1735.

[12] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 28–38. http://sites.computer.org/debull/A15dec/p28.pdf.

[13] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[14] Xilun Chen, K. Selçuk Candan, and Maria Luisa Sapino. 2018. Ims-dtm: Incremental multi-scale dynamic topic models. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 5078–5085.

[15] Jonathan de Andrade Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André Carlos Ponce de Leon Ferreira de Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Survey* 46, 1 (2013), 13:1–13:31. DOI: https://doi.org/10.1145/2522968.2522981

[16] Matthew Hoffman, Francis R. Bach, and David M. Blei. 2010. Online learning for latent dirichlet allocation. In *Proceedings of the Advances in Neural Information Processing Systems*. 856–864.

[17] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *The Journal of Machine Learning Research* 14, 1 (2013), 1303–1347.

[18] Tomoharu Iwata, Takeshi Yamada, Yasushi Sakurai, and Naonori Ueda. 2012. Sequential modeling of topic dynamics with multiple timescales. *ACM Transactions on Knowledge Discovery from Data* 5, 4 (2012), 1–27.

[19] Rolf Jagerman, Carsten Eickhoff, and Maarten de Rijke. 2017. Computing web-scale topic models using an asynchronous parameter server. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1337–1340.

[20] Jon Kleinberg. 2003. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery* 7, 4 (2003), 373–397.

[21] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, CO, .* 583–598.

[22] Shangsong Liang, Emine Yilmaz, and Evangelos Kanoulas. 2016. Dynamic clustering of streaming short documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 995–1004.

[23] Xiaosheng Liu, Jia Zeng, Xi Yang, Jianfeng Yan, and Qiang Yang. 2015. Scalable parallel EM algorithms for latent dirichlet allocation in multi-core systems. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 669–679.

[24] James McInerney, Rajesh Ranganath, and David Blei. 2015. The population posterior and Bayesian modeling on streams. In *Proceedings of the Advances in Neural Information Processing Systems*. 1153–1161.
[25] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y. Arcas. 2016. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS'17)*, Vol. 54. 1273–1282.
[26] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zeda, Matei Zaharia, and Ameet Talwalkar. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.
[27] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. 2015. A survey on data stream clustering and classification. *Knowledge and Information Systems* 45, 3 (2015), 535–569.
[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems*. 8024–8035.
[29] Zhuolin Qiu, Bin Wu, Bai Wang, Chuan Shi, and Le Yu. 2014. Collapsed gibbs sampling for latent dirichlet allocation on spark. In *Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, Vol. 36. 17–28.
[30] Rajesh Ranganath, Chong Wang, Blei David, and Eric Xing. 2013. An adaptive learning rate for stochastic variational inference. In *Proceedings of the International Conference on Machine Learning*. 298–306.
[31] Alan Ritter, Oren Etzioni, and Sam Clark. 2012. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1104–1112.
[32] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.
[33] Lucas Theis and Matt Hoffman. 2015. A trust-region method for stochastic variational inference with applications to streaming data. In *Proceedings of the International Conference on Machine Learning*. 2503–2511.
[34] Jianyu Wang and Gauri Joshi. 2018. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. In *Proceedings of Machine Learning and Systems 2019 (MLSys'19)*, Ameet Talwalkar, Virginia Smith and Matei Zaharia (Eds.). https://proceedings.mlsys.org/book/257.pdf.
[35] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. 2016. Topicsketch: Real-time bursty topic detection from twitter. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2216–2229.
[36] Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.
[37] Hsiang-Fu Yu, Cho-Jui Hsieh, Hyokun Yun, S. V. N. Vishwanathan, and Inderjit S. Dhillon. 2015. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1340–1350.
[38] Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. 2015. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1351–1361.
[39] Lele Yut, Ce Zhang, Yingxia Shao, and Bin Cui. 2017. LDA* a robust and large-scale topic modeling system. In *Proceedings of the VLDB Endowment*. 1406–1417.
[40] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. 423–438.
[41] Manzil Zaheer, Michael Wick, Jean-Baptiste Tristan, Alex Smola, and Guy Steele. 2016. Exponential stochastic cellular automata for massively parallel inference. In *Proceedings of the Artificial Intelligence and Statistics*. 966–975.
[42] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. 2018. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 8 (2018), 2008–2026.
[43] Kuo Zhang, Salem Alqahtani, and Murat Demirbas. 2017. A comparison of distributed machine learning platforms. In *Proceedings of the 26th International Conference on Computer Communication and Networks*. 1–9.

Received July 2020; revised January 2021; accepted February 2021