Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)

Fast and Fine-grained Autoscaler for Streaming Jobs with Reinforcement Learning

Mingzhe Xing[1] , **Hangyu Mao**[2] and **Zhen Xiao**[1*]
[1]School of Computer Science, Peking University
[2]Huawei Noah's Ark Lab
mzxing@stu.pku.edu.cn, maohangyu1@huawei.com, xiaozhen@pku.edu.cn

Abstract

On computing clusters, the *autoscaler* is responsible for allocating resources for *jobs* or fine-grained *tasks* to ensure their Quality of Service. Due to a more precise resource management, fine-grained autoscaling can generally achieve better performance. However, the fine-grained autoscaling for *streaming jobs* needs intensive computation to model the complicated running states of tasks, and has not been adequately studied previously. In this paper, we propose a novel fine-grained autoscaler for streaming jobs based on reinforcement learning. We first organize the running states of streaming jobs as *spatio-temporal graphs*. To efficiently make autoscaling decisions, we propose a Neural Variational Subgraph Sampler to sample spatio-temporal subgraphs. Furthermore, we propose a mutual-information-based objective function to *explicitly* guide the sampler to extract more representative subgraphs. After that, the autoscaler makes decisions based on the learned subgraph representations. Experiments conducted on real-world datasets demonstrate the superiority of our method over six competitive baselines.

1 Introduction

At present, massive streaming data, *e.g.,* video stream, is generated incessantly on service-based applications, and processed by *streaming jobs* [Sun *et al.*, 2019]. Each job is comprised of multiple *tasks* that execute specific computing operations. In order to minimize response latency and to improve the usage efficiency of computing resources, it is critical to design an *autoscaler* [Nguyen *et al.*, 2020], which is responsible for allocating computing resources to jobs or fine-grained tasks, namely *job-level* and *task-level* autoscaling, respectively. Previous work has demonstrated that fine-grained resource management can generally achieve performance improvement in various computing scenarios due to a more precise resource allocation, *e.g.,* 11-x faster execution speed for web services [Qiu *et al.*, 2020] and 35% gain on GPU utilization [Yu *et al.*, 2018]. It is valuable and promising to design an effective task-level autoscaler for streaming jobs.

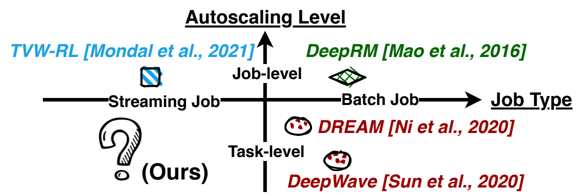


Figure 1: The categories of RL-based autoscalers, which have shown their superiority over heuristic-based methods in previous work.

Designing an optimal autoscaler is known to be NP-hard [Garí *et al.*, 2021]. An easy approach is to use heuristics, namely *heuristic-based autoscalers* [Verma *et al.*, 2015]. They are typically based on heuristic rules meticulously tuned by experts, which is not only a time-consuming process but also subject to human cognitive bias. To remedy these drawbacks, another line of work [Mao *et al.*, 2016] formulates the autoscaling process as Markov Decision Process (MDP), and adopt Reinforcement Learning (RL) to train the autoscalers, namely *RL-based autoscalers*.

Despite the superiority of RL-based autoscalers, how to autoscale streaming jobs at task-level with RL remains a challenging problem and has not been adequately studied. As shown in Fig. 1, DeepRM [Mao *et al.*, 2016], DREAM [Ni *et al.*, 2020] and DeepWave [Sun *et al.*, 2020] used RL to autoscale *batch jobs* that process deterministic size of input data and cannot deal with streaming jobs with *time-variant workloads*. Although TVW-RL [Mondal *et al.*, 2021] considered the temporal patterns of dynamic workloads, it focused on job-level autoscaling rather than task-level, which neglected the topological task dependencies within a job and thus caused poor performance as shown in the experiments. Another obstacle is the *large temporal dimension* issue. Typically, the streaming jobs will be running online for months or even years [Hueske and Kalavri, 2019] and produce massive records of job states. It brings heavy computation overhead to model the recorded fine-grained task states, which will harm the autoscaling efficiency of *time-critical* stream computing.

To address the above issues, we propose a novel ta**S**k-level a**U**toscaler for st**R**eaming jobs with reinforcement l**E**arning (named SURE). First, we give an MDP formulation that can accurately describe the autoscaling process. More

*Corresponding author.

specifically, as a streaming job can normally be encoded as a Directed Acyclic Graph (DAG) [Hueske and Kalavri, 2019], we organize the running states of streaming jobs as *Spatio-Temporal Graphs* (STGs), which take both the topological task dependencies and temporal workloads evolution into consideration and can better model the dynamic task states. Second, due to the large temporal dimension issue, it is inefficient to learn the entire STG. To alleviate this problem, we further design a novel Neural Variational Subgraph Sampler to extract *informative subgraphs*. While the subgraph sampler can be *implicitly* optimized during policy training, we propose an objective function based on mutual information maximization, which can *explicitly* enhance and guide the sampler to extract more representative subgraphs. After that, we leverage Graph Neural Network (GNN) to learn the sampled subgraphs as the representation of the entire STG, which can accelerate the model inference speed significantly. Finally, RL is applied to train the autoscaler.

To our best knowledge, we are the first to utilize RL to learn task-level autoscaler for streaming jobs. Extensive experiments conducted on real-world datasets demonstrate that our method outperforms six baselines. In addition, ablation study, parameter sensitive analysis and visualized case study are provided for better understanding of our work.

2 Related Work

Heuristic-based Autoscaler. HPA [Nguyen *et al.*, 2020] periodically configured the number of replicas based on the monitored metrics, *e.g.,* CPU utilization ratio, and the corresponding desired metric values. Graphene [Grandl *et al.*, 2016] identified tasks that cost a long time to complete, and selected the schedule order with the minimum runing time. Voilà [Fahs *et al.*, 2020] scale-up or scale-down pods by identifing high network latency or overloaded replicas.

Reinforcement-learning-based Autoscaler. Although heuristic-based methods have been widely deployed in industry, they are sub-optimal by nature. To tackle this issue, DeepRM modeled the resource states as bitmap, and applied RL to schedule resources. DeepWeave employed GNN to process DAG information and rewarded solutions with faster Job Completion Time. DREAM designed an Encoder-Decoder framework with GNN and RL to schedule jobs according to their dependencies. However, the aforementioned methods can only schedule batch jobs with static workloads. TVW-RL exploited the temporal patterns of time-varying workloads and used RL to improve the metrics for operational excellence.

Spatio-temporal Graph Modeling. Spatio-temporal graph neural network is becoming growingly important in modeling time evolutionary spatial data. ASTGCN [Guo *et al.*, 2019] consisted of three independent components to model temporal properties, where each component contained the spatio-temporal convolution and attention mechanism to effectively capture the dynamic spatio-temporal correlations. CCRNN [Ye *et al.*, 2021] constructed learnable adjacency matrices in different layers and used a layer-wise coupling mechanism to capture the multi-level spatial dependence and temporal dynamics simultaneously.

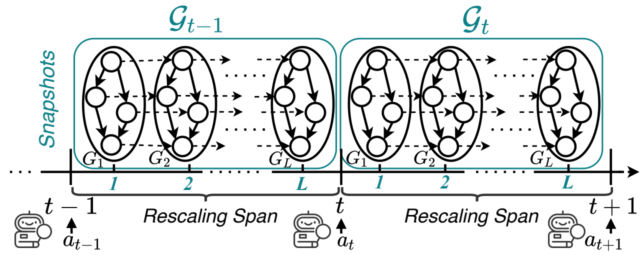


Figure 2: The MDP formulation of autoscaling process of streaming jobs. The running states of jobs (*i.e.,* snapshots) can be organized as spatio-temporal graphs $\mathcal{G}$.

3 Problem Definition

To begin with, we give a formal definition of task-level autoscaling for streaming jobs. Given a streaming job j, it can be abstracted as a DAG, denoted as $\langle \mathcal{V}, \mathcal{E}, \mathcal{P} \rangle$, where $\mathcal{V}$ is the node set and each node denotes a task in job j, and $\mathcal{E}$ represents the dependencies between tasks, and $\mathcal{P}$ denotes how many units of resources are allocated to task nodes, namely *parallelism*. Job j continuously receives and processes *time-varying* size of input data, *e.g.,* video stream. As shown in Fig. 2, for every L minutes (named *rescaling span*), the autoscaler needs to allocate more resources (*scale-up parallelism*) for task nodes if the workloads increase, and release free resources (*scale-down parallelism*) for tasks with low workloads. In the meantime, a monitor records the job DAG states every minute, which can produce L *state snapshots* $[G_1, G_2, \ldots, G_L]$ during a rescaling span, where G_l is the state of job DAG at the l-th minute in rescaling span. In G_l, the i-th task node, denoted as $v_{i,l}$, is associated with a feature vector $\boldsymbol{s}_{i,l} \in \mathbb{R}^D$, where the elements in $\boldsymbol{s}_{i,l}$ are system metrics recorded by monitor, including average and summation of received data size, CPU usage, parallelism and average computation time of $v_{i,l}$, and D is the feature dimension. There are two typical performance metrics for streaming jobs, namely *latency* and *resource utilization ratio*. Specifically, the latency is defined as the duration of a unit of streaming data completely processed by the job, and the resource utilization ratio denotes the proportion of resource usage.

In order to train the autoscaler with RL, we use a tuple $\langle State, Action, Transition, Reward \rangle$ to formulate the autoscaling process as Markov Decision Process (MDP) that can accurately describe the dynamic task states based on the above notations. This MDP tuple can be specified as follows:

• **State**. At the t-th step, by connecting the consecutive L snapshots recorded from the (t-1)-th to t-th step, we can build a *spatio-temporal graph* $\mathcal{G}_{t-1}$ as illustrated in Fig. 2, where each node $v_{i,l}$ is associated with feature vector $s_{i,l}$.

• **Action**. Action $a_{i,t}$, a decimal in $(0.0, 2.0]$, is defined as the ratio between desired parallelism $p_{i,t}$ and current parallelism $p_{i,t-1}$ of task node v_i. This is based on an empirical setting [Taherizadeh and Grobelnik, 2020] that, for each rescaling operation, the parallelism value cannot be reduced to zero or scaled up to more than twice of previous value.

• **Transition**. Given the autoscaling decision $a_{i,t}$ conditioned on $\mathcal{G}_{t-1}$, task node v_i changes its parallelism to $p_{i,t}$. The streaming job then continues running for L minutes, and generates a spatio-temporal graph $\mathcal{G}_t$ as the new state.

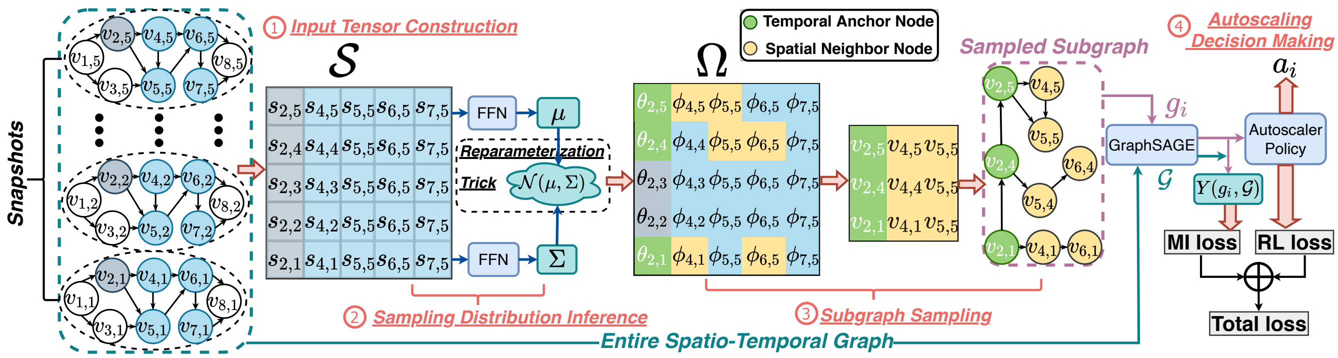


Figure 3: The overall architecture of our proposed approach. It shows an example to sample a subgraph for task node v_2, and then make autoscaling decision for this task node. L, K, k_1 and k_2 are set as 5, 4, 3 and 2 in this example. "FFN" denotes the feed forward network. The steps labeled with ①, ②, ③ and ④ correspond to the four steps introduced in Section 4.1.

• **Reward**. The reward r_t is designed to *minimize* latency l_t and *maximize* resource utilization ratio u_t:

$$r_t = -\lambda l_t + (1 - \lambda) u_t, \tag{1}$$

where $\lambda \in [0, 1]$ controls the importance of each metric and can be tuned according to specific jobs.

With the above MDP formulation, the autoscaler can make decisions based on the topological task dependencies and temporal workloads dynamics implied in the spatio-temporal graphs. However, since the rescaling span L is usually large for streaming jobs [Nguyen *et al.*, 2020], *i.e.,* the *large temporal dimension issue*, the recorded job snapshots are massive, which makes the spatio-temporal graph too huge to be directly learned with conventional *Spatio-Temporal GNNs* [Ye *et al.*, 2021], and brings heavy computation overhead for state modeling and decision-making. Therefore, we need to design a graph learning method, which can accelerate the decision inference speed without performance degradation.

4 Methodology

In this section, we present the learning details of our approach. First, we propose a novel Neural Variational Subgraph Sampler. By learning the sampled informative subgraphs, it can reduce the time cost of autoscaling decision-making. Furthermore, in order to explicitly extract more representative subgraphs, we propose an objective function deduced by mutual information maximization. Finally, we employ RL to train the autoscaler policy.

4.1 Neural Variational Subgraph Sampler

In this part, we aim to tackle the *large temporal dimension issue* and effectively learn the task node representations from giant Spatio-Temporal Graph (STG). Specifically, for each task node in a job DAG, we propose to sample a subgraph from STG, which maintains the most informative features of this task node in STG. By only learning the subgraph instead of the entire STG, it can greatly save the time cost of decision making. Huang and Zitnik also have proved that the information loss of operating GNN on a subgraph rather than the entire graph is bounded by an exponentially decaying term with respect to the node number of subgraph.

More specifically, motivated by the weighted video stream sampling models [Zhi *et al.*, 2021], we assume that there is an underlying *importance weights distribution* of snapshots for each task node along the temporal dimension, and only a subset of snapshots is salient and valuable. Analogously, from the spatial perspective, we also assume that only a subset of spatial neighbors is most relevant with a specific node inspired by Graph Attention Network [Veličković *et al.*, 2018]. Aside from lower computation cost, another benefit of introducing the weighted sampling mechanism is to reduce the noise or irrelevant features that could lead to performance degradation. Based on the above assumptions, we design a novel Neural Variational Subgraph Sampling (named NVSS) strategy, consisting of the following four steps.

① **Input Tensor Construction.** Our goal is to derive the temporal and spatial sampling probability distributions $p(\theta_i|\mathcal{S}_i)$ and $p(\phi_i|\mathcal{S}_i)$, where θ_i is the importance weights along temporal dimension of task node v_i, and ϕ_i^l is the spatial sampling weights of v_i at the l-th snapshot, and $\mathcal{S}_i$ denotes the running states of v_i. First, for node v_i, we build an input tensor $\mathcal{S}_i \in \mathbb{R}^{L\times(1+K)\times D}$ from the L recorded snapshots, where $K = |\mathcal{N}_c(v_i)|$ is the number of c-hop neighbor nodes of v_i. As shown in Fig. 3, the first column (in gray) of $\mathcal{S}_i$ is the feature vectors $[\boldsymbol{s}_{i,1}, \boldsymbol{s}_{i,2}, \dots, \boldsymbol{s}_{i,L}]^T$ of nodes $\{v_{i,1}, v_{i,2}, \dots, v_{i,L}\}$, and the last K elements (in blue) at the l-th row are the feature vectors of the c-hop neighbors of $v_{i,l}$ (for brevity, we hide the feature dimension D in Fig. 3, and set $L = 5$ and $K = 4$, so the shape of $\mathcal{S}_i$ is $5 \times (1 + 4)$).

② **Sampling Distribution Inference.** To capture the underlying correlation of spatial and temporal domains, we aim to learn the joint spatial and temporal importance weights distribution $p(\Omega_i|\mathcal{S}_i) = p(\theta_i, \phi_i|\mathcal{S}_i)$ instead of learning $p(\theta_i|\mathcal{S}_i)$ and $p(\phi_i|\mathcal{S}_i)$ separately. In specific, we design a variational inference network to obtain $p(\Omega_i|\mathcal{S}_i)$. Here, we assume $p(\Omega_i|\mathcal{S}_i)$ follows Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_i)$, and derive the mean $\mu_i \in \mathbb{R}^{L\times(1+K)}$ and co-variance $\Sigma_i \in \mathbb{R}^{L\times(1+K)}$:

$$\mu_i = \sigma(\mathbf{W}_1^T \mathcal{S}_i), \quad \Sigma_i = \sigma(\mathbf{W}_2^T \mathcal{S}_i),$$

where σ is activation function, and $\mathbf{W}_1$ and $\mathbf{W}_2$ are learnable parameters. Then, the joint spatio-temporal sampling distribution $\Omega_i \in \mathbb{R}^{L\times(1+K)}$ can be drawn from $\mathcal{N}(\mu_i, \Sigma_i)$. We can obtain the θ_i and ϕ_i^l from Ω_i as follows:

$$\theta_i = (\Omega_{m,n})_{\substack{1\le m\le L,\\ n=1}}, \quad \phi_i^l = (\Omega_{m,n})_{\substack{m=l\\ 2\le n\le K+1}},$$

where $\theta_i \in \mathbb{R}^L$ and $\phi_i^l \in \mathbb{R}^K$ are the first column and the last K elements at the l-th row from Ω_i, respectively.

③ **Subgraph Sampling.** Next, we sample a subgraph for node v_i using θ_i and ϕ_i. First, k_1 informative snapshots can be sampled from $Multinomial(\theta_i)$ for node v_i. We name the v_i in the corresponding sampled snapshots as *temporal anchor nodes*, *e.g.,* $v_{2,1}$, $v_{2,4}$ and $v_{2,5}$ (green nodes) in Fig. 3. Second, for each temporal anchor node $v_{i,l}$, k_2 *spatial neighbor nodes* (yellow nodes) can be sampled from $Multinomial(\phi_i^l)$ in the l-th snapshot, where the spatial neighbor nodes are from the c-hop neighbors set of v_i, *i.e.,* $\mathcal{N}_c(v_i)$. In this way, we can obtain the sampled temporal anchor nodes, and the corresponding spatial neighbor nodes. By connecting temporal anchor nodes as depicted in Fig. 3, we can reconstruct the sampled subgraph g_i for task node v_i. Under this sampling procedure, the marginal likelihood of g_i is:

$$p(g_i|\mathcal{S}_i) = \prod_{l=1}^{k_1}\prod_{s=1}^{k_2} p(v_{l_s}|\phi_i^l)p(v_l|\theta_i)p(\theta_i,\phi_i|\mathcal{S}_i)), \quad (2)$$

where v_{l_s} and v_l are the sampled spatial neighbor and temporal anchor nodes, respectively.

Note that the sampling processes include drawing Ω_i from $\mathcal{N}(\mu_i, \Sigma_i)$ and sampling nodes from $p(\theta_i)$ and $p(\phi_i)$, but they are undifferentiable, making back-propagation inapplicable. To solve this issue, we adopt the Reparameterization Trick [Kingma *et al.*, 2015] that can be specified as follows:

$$\boldsymbol{x} \sim \mathcal{N}(0, diag(\boldsymbol{I})) \quad \boldsymbol{y} \sim Gumbel(0,1)$$
$$\Omega_i = \log(\boldsymbol{x} \times \Sigma_i + \mu_i) + \boldsymbol{y},$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sampled random variables from Gaussian and Gumbel distribution respectively, and $diag(\boldsymbol{I})$ is a diagonal matrix with all-one diagonal elements. In this way, we can transfer the undifferentiable factor to $\boldsymbol{x}$ and $\boldsymbol{y}$, while μ_i and Σ_i are differentiable and can be optimized during training.

④ **Autoscaling Decision Making.** After obtaining the sampled subgraph g_i, we utilize GraphSAGE [Hamilton *et al.*, 2017] to learn subgraph representation as the state of task node v_i, which can save considerable amount of state modeling time compared with learning the entire STG (detailed time complexity analysis can be found in *Appendix B*). The message passing process of GraphSAGE can be specified as:

$$\boldsymbol{h}_i^e = \sigma(\mathbf{W}_3^e[\boldsymbol{h}_i^{e-1}; \frac{\sum_{u\in\mathcal{M}(i)}\boldsymbol{h}_u^{e-1}}{|\mathcal{M}(i)|}]),$$

where $\mathbf{W}_3^e$ is the learnable parameter at the e-th layer of GraphSAGE, and $\mathcal{M}(i)$ is the neighbor nodes set of v_i. The input $\boldsymbol{h}^1$ of the first layer of GraphSAGE is the node features $\{\boldsymbol{s_u}\}_{u\in g_i}$. Recall that subgraph g_i is sampled with respect to node v_i, we can regard $\boldsymbol{h}_{g_i} = \text{Readout}(\boldsymbol{h}_u^E|u \in g_i)$ as the representation of v_i, where the Readout function is average pooling and E is the number of GraphSAGE layers.

Finally, a linear layer acts as the agent to make autoscaling decision for v_i based on the state representation $\boldsymbol{h}_{g_i}$:

$$a_i = \sigma(\mathbf{W}_4^T\boldsymbol{h}_{g_i}), \quad (3)$$

where a_i is the action decision of autoscaling (introduced in Section 2), and $\mathbf{W}_4$ is a learnable parameter.

4.2 Enhancing NVSS via Mutual Information

In the previous section, NVSS can be *implicitly* learned to sample informative spatio-temporal subgraphs, due to the existing of reward signal from RL. In this section, we *explicitly* encourage the sampled subgraphs to reveal the most representative topological information of the entire STG, and enhance NVSS to extract more representative subgraphs.

Specifically, we would like to minimize the semantic divergence between the sampled subgraph g_i and the entire STG $\mathcal{G}$. Since the Mutual Information (MI) [Kraskov *et al.*, 2004] measures the mutual dependence between two variables, and a larger MI means that the two variables are more correlated, we propose an objective function from the MI perspective as follows (we omit the subscript of g_i for concision):

$$\max I\big(f(g), f(\mathcal{G})\big) = H\big(f(g)\big) - H\big(f(g)|f(\mathcal{G})\big), \quad (4)$$

where H denotes the entropy of the given distribution, and f denotes the GraphSAGE that extracts features for graphs.

With the likelihood distribution of subgraph g (Eq. 2), the lower bound of objective function (Eq. 4) can be derived as follows (please see *Appendix A* for detailed derivation):

$$Y(g,\mathcal{G}) = log\mathbb{E}_{g\sim p(g|\mathcal{S})}I\big(f(g), f(\mathcal{G})\big) \quad (5)$$
$$\geq \sum_g \Big(-\mathcal{KL}\big(q(\Omega|\mathcal{S})||p(\Omega|g,\mathcal{S})\big)$$
$$-\big(\frac{1}{2}log|\Sigma| + (\Omega-\mu)^T\Sigma^{-1}(\Omega-\mu) + CE(\Omega,\hat{\Omega})\big)$$
$$+\mathbb{E}_q logI\big(f(g), f(\mathcal{G})\big)\Big), \quad (6)$$

where $\mathcal{KL}$ represents KL-divergence [Anzai, 2012], and CE denotes Cross Entropy loss, and $\hat{\Omega}$ is the joint distribution of actually sampled temporal anchor and spatial neighbor nodes.

The objective function (Eq. 6) includes three parts, *i.e.,* KL-divergence, log-likelihood of prior distributions and log of MI, which are intuitive. The KL-divergence term attempts to minimize the distance between the estimated distribution q and the true distribution p. The second term is regularization term of prior distributions. The log of MI ensures that the sampled subgraphs contain the most representative information of the entire STG. The first and second terms in Eq. 6 are easy to compute. As for the MI term, we adopt the Jensen-Shannon MI estimator [Nowozin *et al.*, 2016] as follows:

$$I\big(f(g), f(\mathcal{G})\big) = \mathbb{E}_p[-\text{sp}(D_w\big(f(g), f(\mathcal{G})\big))]$$
$$-\mathbb{E}_{\hat{p}}[\text{sp}(D_w\big(f(g')\big), f(\mathcal{G}))],$$

where D_w is a discriminator that takes a subgraph and STG embeddings pair as input and determines whether the subgraph is sampled from the STG, and $\text{sp}(z) = \log(1+e^z)$ is the softplus function, and g' is negative sampled subgraph from $\hat{p} = p$. In practice, we generate negative subgraphs by using all possible combinations of STG and subgraph embeddings across all graph instances in a mini-batch.

Indeed, there have been a few studies that used subgraph sampling and mutual information techniques simultaneously, however, our method has two major differences. First, NVSS can capture the underlying correlation of spatial and temporal domains, and be end-to-end optimized, while the samplers in previous work are heuristic-based [Sun *et al.*, 2021] or cannot perform sampling for a specific node along the temporal

		Small-1	Small-2	Medium-1	Medium-2	Large-1	Large-2	Average
Heuristic-based	HPA	-0.17	1.16	-2.69	-0.90	-1.28	-2.35	-1.04
RL-based	DeepWave	-2.77	-1.23	0.16	-0.97	0.69	0.32	-0.63
	DREAM	0.50	-0.23	0.23	-0.11	0.92	-1.14	0.03
	TVW-RL	0.26	0.66	0.08	-0.40	0.95	0.85	0.40
Spatial-temporal GNN	ASTGCN	0.26	-0.66	0.36	1.09	-1.24	0.29	0.02
	CCRNN	0.48	0.97	0.24	1.12	-0.57	0.46	0.45
Ours	SURE	**0.52**	**1.41**	**1.19**	**1.81**	**1.02**	**0.95**	**1.15**

Table 1: Performance comparison with baselines on *Small*, *Medium* and *Large* job settings, respectively. The best, second best and third best results are in bold, underline and gray cell, respectively.

dimension [Yu *et al.*, 2020], which cannot be applied in this scenario and compared with our method. Second, we seek for a seamless integration of the subgraph sampling and mutual information rather than a simple combination of the two techniques. With the derived objective function (Eq. 6), the two parts can be jointly trained and mutually optimized. Moreover, the objective function is also intuitive and interpretable.

4.3 Training with Reinforcement Learning

Here, we use RL to optimize the autoscaler policy. The expected sum of rewards is $R = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where π represents the policy that makes autoscaling decisions with Eq. 3, and $\gamma \in (0, 1]$ is a discount factor that reduces future rewards relative to current reward, and r_t is the reward function as defined in Section 2. The objective function that needs to be maximized for optimizing the autoscaler policy is: $J(\psi) = \frac{1}{N}\sum_{n=1}^{N} log\pi_\psi R$, where ψ is the model parameters. Combined with the MI-based objective function (Eq. 6), the total loss of our model is $\mathcal{L} = -J - \lambda_1 \sum_{i=1}^{|V|} Y_i$, where λ_1 is a hyperparameter that adjusts the weight of the MI-based loss. More learning details can be found in *Appendix C*.

5 Experiments

5.1 Experimental Setup

Simulation Environment. Following the streaming job execution logic described in Section 2, we implement a simulated computing system for streaming jobs and register a monitor for each job to record its state snapshots periodically.

Dataset. We use Clarknet Trace [1] as workloads, which describes the number of HTTP requests to the servers recorded in 20,000 minutes. The workloads are highly varying in time, and show periodicity characteristics. As for computing jobs, since we focus on long-running streaming jobs, we only keep the jobs in Alibaba Cluster Dataset [2] that were running for more than 2,000 minutes following Mondal *et al.*. The details of datasets can be found in *Appendix D.1*.

Experiment Settings. To evaluate the performance of our method on jobs with different task numbers, we randomly sample six jobs and divide them into three sets, namely *Small*, *Medium* and *Large*. More specifically, the task numbers of *Small-1*, *Small-2*, *Medium-1*, *Medium-2*, *Large-1* and *Large-2* are 6, 16, 25, 32, 40 and 46, respectively. Each job acts as an individual simulation environment and receives a subsequence of workloads that lasts for 7×24 hours in ClarkNet Trace in each episode. For each job, we first train an autoscaler and then use it to perform testing on this job for ten times and take the average reward as the final result.

Baselines. We compare our method with baselines from three categories: (1) heuristic-based autoscaler, *i.e.,* HPA; (2) RL-based autoscaler, *i.e.,* DeepWave, DREAM and TVW-RL; and (3) spatio-temporal GNN, *i.e.,* ASTGCN and CCRNN. Due to space limitation, we skip their details that have been presented in related work. For fair comparisons, we use the same features for all baselines, and make slight modifications for them to fit for our scenario. Following DREAM, DeepWave and TVW-RL, we use REINFORCE [Williams, 1992] to train our autoscaler. The details of parameter settings and modifications of all compared methods are in *Appendix D.2*. The code and *Appendix* are available at https://github.com/xmzzyo/sure.

5.2 Performance Comparison

We present the standardized rewards of all baselines and our method on *Small*, *Medium*, *Large* job settings in Table 1.

In general, RL-based autoscalers perform better than HPA on most jobs, because their strategies can be optimized regarding specific job environments, while HPA uses heuristic rules and cannot be well adapted to different jobs and dynamic workloads. TVW-RL performs the best among RL-based methods on average, since it can model the temporal patterns of workloads, which is critical when autoscaling streaming jobs. Among all the baselines, CCRNN achieves the best performance on average. It uses a layer-wise coupling mechanism to capture the multi-level dependence of temporal and spatial domains. Our method consistently outperforms all the competitors on all jobs. It strongly supports our claim that the proposed subgraph sampling strategy can extract representative subgraphs, which imply the joint spatial and temporal correlations of the entire STG. By only learning the subgraphs, it can derive effective task representations and make good decisions. Moreover, our sampling mechanism can also alleviate the interference of redundancy information or noise, which can benefit the model performance.

5.3 Ablation Study

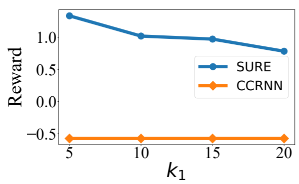
To examine the effects of different modules, three variants of our method are compared, including: (A) **w/o NVSS** denotes using an uniformly random sampler instead of NVSS; (B) **w/o**

[1] ftp://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

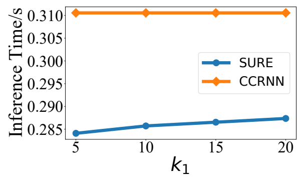
[2] https://github.com/alibaba/clusterdata

	Small-1	Small-2	Medium-1	Medium-2	Large-1	Large-2
w/o NVSS	0.46	-1.30	0.21	-0.82	-1.27	0.29
w/o MI	0.47	-0.78	0.22	-0.82	0.78	0.33
SURE	**0.52**	**1.41**	**1.19**	**1.81**	**1.02**	**0.95**

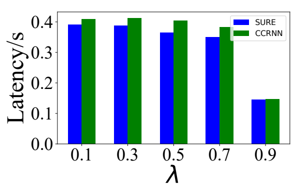
Table 2: Ablation study of our model. The best results are in bold.



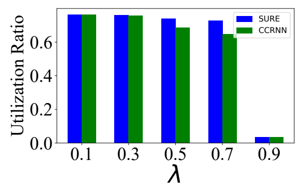
(a) Reward.



(b) Inference time (in second).



(c) Latency (lower is better).



(d) Utilization Ratio (higher is better).

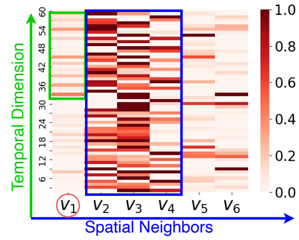
Figure 4: Sensitivity analysis by varying k_1 and λ for Large-1 job.

MI denotes using only NVSS without mutual-information-based objective function (Eq. 6); (C) **SURE** denotes our complete model. Table 2 indicates that the random sampling strategy does not perform well. Nevertheless, with our subgraph sampling strategy NVSS, it can extract informative subgraphs that contribute to a higher final reward. By incorporating our proposed objective function based on MI, it can lead to a great performance improvement, which proves that this objective function can guide the NVSS to extract more representative subgraphs and learn effective task representations.

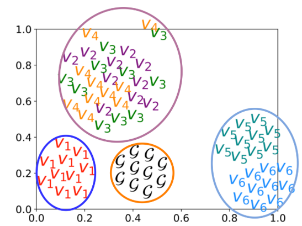
5.4 Performance Tuning

One major contribution of this paper is the subgraph sampling strategy, which can reduce the time cost of model inference, while achieving superior performance. In this part, we report the inference time and reward influenced by the number of sampled snapshots (*i.e.,* k_1). In addition, by varying λ in Eq. 1, we also examine the sensitivity of metrics that compose the reward, *i.e.,* latency and utilization ratio. Due to space limitation, we only present the results on Large-1 job (the results on the other jobs are similar and can be found in *Appendix D.3*), and incorporate the best baseline CCRNN from Table 1 for comparison.

Fig. 4(a) shows that our model consistently outperforms the best baseline, and the reward declines when k_1 increases. A possible reason is that redundant snapshots may introduce noise or irrelevant information that lead to performance degradation for Large-1 job. Fig. 4(b) indicates that the inference time of our model is much less than CCRNN and shows a rising trend as k_1 increases, since learning more snapshots would bring more computation overhead. From



(a) Sampling distribution Ω.



(b) Graph embeddings.

Figure 5: Case study based on Small-1 job.

Fig. 4(c) and 4(d), we can see that our approach can achieve lower latency and competitive utilization efficiency on most λ settings. Note that we cannot always have improvements on both metrics, since a lower latency would generally cost more or even redundant resources. Nevertheless, the comprehensive performances of our model, *i.e.,* the reward and inference time, are consistently better than the best baseline.

5.5 Case Study

In this part, we show qualitative cases based on Small-1 job to give intuitive impressions of our model. The DAG structure of Small-1 job can be found in *Appendix D.4*

Fig. 5(a) presents the heatmap of sampling distribution Ω of task node v_1. From the first column of Ω (*i.e.,* θ), we can observe that the most recent snapshots (framed in green) are more likely to be sampled along temporal dimension, which is reasonable and intuitive since they are more related to the states at the next step. Besides, the nearest (1-hop) neighbor nodes of v_1, *i.e.,* v_2, v_3 and v_4 (framed in blue), have larger probabilities to be sampled, since they are more topological relevant to v_1. In Fig. 5(b), we use t-SNE to visualize the embeddings of subgraphs sampled for task nodes and the entire STG in ten autoscaling steps. As depicted, the embeddings are grouped into four clusters. The v_2, v_3 and v_4 are grouped together as they are 1-hop neighbors of v_1, and v_5 and v_6 are 2-hop and 3-hop neighbors of v_1. It demonstrates that our graph embeddings can effectively present the topological features of nodes, and the subgraph embeddings are centralized and closed with the embedding of the entire STG $\mathcal{G}$.

6 Conclusion

In this paper, we propose a novel approach to autoscale streaming jobs at task-level with reinforcement learning. We first organize the job states as spatio-temporal graphs and give a formal MDP formulation of autoscaling process. To efficiently learn the giant spatio-temporal graphs, we design a Neural Variational Subgraph Sampler, which can greatly save the graph learning time. Furthermore, we propose an objective function based on mutual information to guide the sampler to extract more representative subgraphs. Our experiments demonstrate the superior performance and interpretability of our approach. In the future, we will apply our method to solve other classical spatio-temporal graph modeling tasks, such as traffic forecasting and pose detection, which also suffer from the large temporal dimension issue.

Acknowledgments

We thank Jianye Hao, Dong Li, Chen Chen, and the anonymous reviewers for their insightful comments on this paper. This work was supported by the National Natural Science Foundation of China under Grant No. 61872397. Zhen Xiao is the corresponding author.

References

[Anzai, 2012] Yuichiro Anzai. *Pattern recognition and machine learning*. Elsevier, 2012.

[Fahs *et al.*, 2020] Ali J. Fahs, Guillaume Pierre, and Erik Elmroth. Voilà: Tail-latency-aware fog application replicas autoscaler. In *MASCOTS*, 2020.

[Garí *et al.*, 2021] Yisel Garí, David A Monge, Elina Pacini, Cristian Mateos, and Carlos García Garino. Reinforcement learning-based application autoscaling in the cloud: A survey. *Eng. Appl. Artif. Intell.*, 2021.

[Grandl *et al.*, 2016] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. {GRAPHENE}: Packing and dependency-aware scheduling for data-parallel clusters. In *OSDI*, 2016.

[Guo *et al.*, 2019] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proc. of AAAI*, 2019.

[Hamilton *et al.*, 2017] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. of NeuIPS*, 2017.

[Huang and Zitnik, 2020] Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. *NeuIPS*, 2020.

[Hueske and Kalavri, 2019] Fabian Hueske and Vasiliki Kalavri. *Stream processing with Apache Flink: fundamentals, implementation, and operation of streaming applications*. O'Reilly Media, 2019.

[Kingma *et al.*, 2015] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *NeuIPS*, 2015.

[Kraskov *et al.*, 2004] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 2004.

[Mao *et al.*, 2016] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *HotNets*, 2016.

[Mondal *et al.*, 2021] Shanka Subhra Mondal, Nikhil Sheoran, and Subrata Mitra. Scheduling of time-varying workloads using reinforcement learning. In *Proc. of AAAI*, 2021.

[Nguyen *et al.*, 2020] Thanh-Tung Nguyen, Yu-Jin Yeom, Taehong Kim, Dae-Heon Park, and Sehan Kim. Horizontal pod autoscaling in kubernetes for elastic container orchestration. *Sensors*, 2020.

[Ni *et al.*, 2020] Xiang Ni, Jing Li, Mo Yu, Wang Zhou, and Kun-Lung Wu. Generalizable resource allocation in stream processing via deep reinforcement learning. In *Proc. of AAAI*, 2020.

[Nowozin *et al.*, 2016] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Proc. of NeuIPS*, 2016.

[Qiu *et al.*, 2020] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. {FIRM}: An intelligent fine-grained resource management framework for slo-oriented microservices. In *OSDI*, 2020.

[Sun *et al.*, 2019] Guang Sun, YingJie Song, ZiQin Gong, Xiya Zhou, Xinyi Zhou, and YiLin Bi. Survey on streaming data computing system. In *ACM TUR-C*, 2019.

[Sun *et al.*, 2020] Penghao Sun, Zehua Guo, Junchao Wang, Junfei Li, Julong Lan, and Yuxiang Hu. Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling. In *Proc. of IJCAI*, 2020.

[Sun *et al.*, 2021] Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proc. of WWW*, 2021.

[Taherizadeh and Grobelnik, 2020] Salman Taherizadeh and Marko Grobelnik. Key influencing factors of the kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. *Adv. Eng. Softw*, 2020.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Verma *et al.*, 2015] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proc. of EuroSys*, 2015.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[Ye *et al.*, 2021] Junchen Ye, Leilei Sun, Bowen Du, Yanjie Fu, and Hui Xiong. Coupled layer-wise graph convolution for transportation demand prediction. In *Proc. of AAAI*, 2021.

[Yu *et al.*, 2018] Chao Yu, Yuebin Bai, Hailong Yang, Kun Cheng, Yuhao Gu, Zhongzhi Luan, and Depei Qian. Smguard: A flexible and fine-grained resource management framework for gpus. *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[Yu *et al.*, 2020] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. In *Proc. of ICLR*, 2020.

[Zhi *et al.*, 2021] Yuan Zhi, Zhan Tong, Limin Wang, and Gangshan Wu. Mgsampler: An explainable sampling strategy for video action recognition. In *Proc. of ICCV*, 2021.