Received 27 August 2025, accepted 16 September 2025, date of publication 19 September 2025, date of current version 26 September 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3611994

RESEARCH ARTICLE

SearchLight: Neural Architecture Search for Lightweight Spatio-Temporal Graph Neural Networks

HEEYONG YOON[1], JINHONG JUNG[2], KANG-WOOK CHON[3], AND MIN-SOO KIM[4], (Senior Member, IEEE)

[1]Department of Electrical Engineering and Computer Science, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu 42988, South Korea
[2]School of Software, Soongsil University, Seoul 06978, South Korea
[3]School of Computer Science and Engineering, Korea University of Technology and Education (KOREATECH), Cheonan 31253, South Korea
[4]School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea

Corresponding authors: Kang-Wook Chon (kw.chon@koreatech.ac.kr) and Min-Soo Kim (minsoo.k@kaist.ac.kr)

This work was supported by the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT)(IITP-2025-RS-2020-II201795), the National Research Council of Science & Technology (NST) grant by the Korea government (MSIT) (No. GTL24031-000), the MSIT (Ministry of Science and ICT), Korea, under the Convergence security core talent training business support program (IITP-2025-RS-2024-00426853) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation), and the Education and Research Promotion Program of KOREATECH in 2024.

ABSTRACT Spatio-Temporal Graph Neural Networks (STGNNs) are neural network models that integrate spatial information into time series processing, and have been successfully applied in various applications. Although these models have demonstrated strong prediction capabilities, most existing STGNN architectures require a significant memory size and long training times. Some lightweight versions of STGNNs have been proposed, but they still rely on expert-driven manual designs to improve performance, which require implicit domain-specific knowledge that varies across datasets. This design approach limits their adaptability to different application scenarios. To address this limitation, Neural Architecture Search (NAS) has been applied to automate the STGNN design process. However, existing NAS-based approaches prioritize prediction accuracy rather than resource efficiency. As a result, current approaches fail to provide compact model architectures or efficient training and limit their scalability. In this work, we introduce SearchLight, a novel NAS-based STGNN framework to automatically discover lightweight STGNN models while maintaining prediction performance. We set two cells for spatial and temporal operations into two distinct sets to capture spatial and temporal data features better for each cell type of the NAS method. We specialize in cell types for spatial and temporal information so that the model can better capture and combine the intrinsic features of spatial and temporal data. We employ a multi-objective search strategy that optimizes both model compactness and prediction accuracy to enable our method to discover lightweight and accurate STGNN models. Experimental results across several real-world datasets show that SearchLight reduces the model size by an average of $\times 103.0$ and training time by an average of $\times 74.0$, while sacrificing a small amount of prediction performance, an average of 1.6%p, compared to manually designed and existing NAS-based STGNN models.

INDEX TERMS Spatio-temporal graph neural networks, neural architecture search, lightweight model, time-series data, multi-objective optimization.

I. INTRODUCTION

Spatio-Temporal Graph Neural Networks (STGNNs) are a group of neural network models utilizing spatial information to handle time series data, which considers spatial

The associate editor coordinating the review of this manuscript and approving it for publication was Fahmi Khalifa.

relationships as graph edges and temporal data as the vertex features. These models have been successfully applied in prediction tasks of various domains such as vehicle traffic network [1], [2], [3], [4], [5], [6], [7], [8], [9], video understanding [2], [10], epidemics [11], weather forecasting [12], human pose detection [13], brain signal detection [14], and environmental monitoring [15]. By its hybrid structure, STGNN has outperformed classical time series models such as ARIMA [16] and VAR [17].

Although active research for enhancing prediction accuracy has appeared in diverse domains, relatively little attention has been paid to the computational efficiency of STGNN model design. Currently, many STGNN studies focus on designing models with large parameters and long training time to achieve the lowest prediction error. These characteristics limit their applicability in real-world scenarios where computational resources are constrained or shared among multiple tasks. This resource-intensive trend can limit the application of STGNN in real-world scenarios, especially in limited computing resources such as a processor scheduled by several tasks in autonomous vehicles [18] or small devices for constructing Internet of Things (IoT) [19]. Therefore, a lightweight model is required to achieve a minimal memory footprint and maintain its prediction accuracy so as not to disrupt the requested tasks.

For this resource problem, some studies proposed lightweight STGNN models [20], [21], [22], [23], yet they designed model architecture manually. The manual approach to designing architecture often becomes time-consuming and relies heavily on the expert's intuition. Additionally, these handcrafted models are usually evaluated on a small number of datasets due to their limited development time, which can lead to architectural overfitting and poor generalization. Furthermore, since spatio-temporal data has nonlinear and highly correlated characteristics and shows different statistical features depending on the domain, a manually designed single architecture can easily fail to capture data patterns across different applications. Despite some researches have introduced Neural Architecture Search (NAS) against designing STGNN models manually [24], [25], [26], [27], [28], [29], they have primarily focused on reducing prediction error, with little attention to efficiency in terms of training time or memory consumption.

To address these limitations, we propose SearchLight, a novel NAS framework, which aims to discover a lightweight STGNN model automatically. Our approach utilizes NAS to automate the design process and prevent manual intervention requiring expert trial and error. We choose one of the NAS methods, Differentiable Architecture Search (DARTS) [30], to find lightweight and resource-efficient structures to reduce model development time quickly. We divide cell types into S-cell and T-cell, which use a disjoint set of operations to accommodate better spatial and temporal features of input data in finding data-adaptive architectures. S-cell is a collection of graph-based layers to extract spatial patterns, while T-cell has neural networks for sequence modeling to capture temporal dependencies. Additionally, we propose an objective function to find lightweight architecture, consisting of the prediction error and the number of model parameters. The objective function encourages the search algorithm to prefer an architecture that targets compact-sized models yet maintains acceptable prediction accuracy. The key differences between our method and existing methods are summarized in Table 1.

TABLE 1. Comparison between our work and other approaches.

	Normal STGNNs	Lightweight STGNNs	STGNNs from NAS	**SearchLight (our work)**
Automated Search	No	No	Yes	**Yes**
Lightweight Model	No	Yes	No	**Yes**
Training Time	Long	Short	Long	**Short**
Model Complexity	Large	Small	Large	**Small**

Our main contributions are as follows:

- We propose SearchLight, a novel NAS framework designed to discover lightweight GNN-based model for spatio-temporal graphs. To the best of our knowledge, this is the first attempt to find a lightweight STGNN model automatically.
- We introduce strategies that consider both prediction performance and model size to design a lightweight STGNN model. First, we assign different cell types to handle spatial and temporal features separately, so that the model can better capture the characteristics of spatio-temporal data. Then, we use a multi-objective loss function that optimizes both model compactness and prediction accuracy.
- We demonstrate that SearchLight achieves comparable predictive performance while significantly reducing model complexity (i.e., model parameter size and training time). Through extensive experiments on diverse real-world datasets, we show that the architectures discovered by our approach outperform manually designed models and existing NAS-based STGNNs in parameter count, search time, and training time, without significant loss in accuracy.

Section II introduces prior works on STGNN and NAS related to our study. Section III explains background knowledge for this study. Section IV presents a detailed explanation of our method, SearchLight, and Section V shows evaluation results and analysis of our method with existing competitors. Finally, Section VI concludes this paper.

II. RELATED WORK

This section introduces some of the research closely related to our study. Section II-A presents some of the popular STGNNs, and Section II-B introduces manually designed lightweight STGNNs to reduce memory usage and training time. Section II-C covers recent studies that apply NAS techniques to discover the neural architectures of STGNNs automatically.

A. SPATIO-TEMPORAL GRAPH NEURAL NETWORKS

Many Spatio-Temporal Graph Neural Networks (STGNNs) have been proposed to capture complex spatial dependencies and temporal dynamics in traffic prediction tasks. These models are widely recognized with over a thousand citations each, and they are the foundation for the following lightweight and NAS-based approaches of STGNNs. Adaptive Graph Convolutional Recurrent Network (AGCRN) [31] captures unique spatiotemporal patterns of individual traffic sensors. It employs a Node Adaptive Parameter Learning (NAPL) module to learn sensor-specific parameters and a Data Adaptive Graph Generation (DAGG) module to dynamically generate graph structures during training by creating sensor embeddings. Attention-based Spatial-Temporal Graph Convolutional Network (ASTGCN) [6] learns complex and dynamic spatiotemporal correlations in traffic data. It introduces spatiotemporal attention mechanisms to learn intricate spatial relationships between locations and dynamic temporal dependencies over time. The model extracts spatial features using graph convolution and models temporal dependencies with temporal convolution. Graph WaveNet (GWNet) [32] focuses on capturing latent spatial dependencies and long-term temporal relationships. It identifies latent relationships in sensor signals and uses stacked dilated causal convolutions, inspired by WaveNet [33], to achieve a wide receptive field with few layers for effective temporal modeling.

B. LIGHTWEIGHT SPATIO-TEMPORAL GRAPH NEURAL NETWORKS

While traditional Spatio-Temporal Graph Neural Networks (STGNNs) show high prediction accuracy, their computational and memory requirements often make them difficult to apply in real-world traffic systems. Lightweight STGNN models have been researched to address these issues. These studies can be broadly divided into two main directions: optimizing core computational methods or redesigning the data processing flow and the model structure.

The first approach optimizes core operations and focuses on making complex computational processes more efficient. For example, ST-TIS [20] uses a technique known as region sampling. In this method, a group of closely located sensors and a few representative sensors from distant areas are sampled together rather than all other sensors in the network. This strategy significantly reduces computational complexity while maintaining overall network relationships. Another model, STGDN [21], uses a convolution neural network with a fixed dilation pattern. This method processes past data at fixed intervals (e.g., 5, 10, and 20 minutes prior) to analyze a wide temporal range with less computation. However, these optimization methods have inherent limitations. The sampling method of ST-TIS can miss important sensor information if a critical node is excluded from the sample, such as a randomly occurring traffic jam in a distant area. Furthermore, the fixed pattern of STGDN is effective for predictable and periodic changes like rush hour. However, it can be weak for capturing dynamic dependencies such as sudden accidents or events.

The second approach maximizes efficiency by enhancing the model's structure and data processing flow. BigST [22] utilizes long-term embedding. It analyzes historical data to create a compact representation. The model utilizes this summary during inference to produce results rapidly. This approach is similar to studying for an exam that a student memorizes with a well-organized cheat sheet instead of re-reading the entire textbook. Meanwhile, STLGRU [23] introduces a gating mechanism. These gates distinguish between essential and redundant information within the input data and help focus computational resources on important features. However, such structural innovations also come with trade-offs. The pre-computed summary in BigST is challenging to update in real-time, making it hard to respond quickly to sudden changes. Additionally, for highly irregular or sparse data, the gating mechanism of STLGRU may not correctly distinguish between signals and noise, potentially leading to a decrease in prediction accuracy.

In contrast to these manually designed lightweight models, our approach does not rely on expert-driven heuristics. Instead, we employ NAS to discover compact architectures for each dataset automatically. It enables SearchLight to achieve efficiency without the risk of architectural overfitting, which has bias on specific design choices.

C. NEURAL ARCHITECTURE SEARCH FOR SPATIO-TEMPORAL GRAPH NEURAL NETWORKS

Automatic architecture findings for STGNN have also been studied. These studies aim to replace the complex and challenging manual design process with NAS. There are several categories of NAS methods, such as reinforcement learning–based methods (i.e., NAS-RL), cell-based search methods (i.e., NASNet and its variants), and gradient-based methods (i.e., DARTS), which are explained in detail in Section III-B. All existing NAS-based STGNN studies are based on the DARTS framework due to its efficiency and relatively significantly shorter searching time.

The research can be categorized into two approaches. The first approach focuses on the internal module search technique. For example, AutoCTS [24] employs a hierarchical search method. The micro-search stage designs the optimal internal structure of core ST blocks, and the macro-search stage determines how these ST blocks are interconnected. Similarly, AutoMSNet [25] focuses only on designing the spatial feature extraction module rather than the entire architecture. AutoMSNet automatically combines graph convolution operators with different receptive fields to create a spatial processor for dataset characteristics.

Another major approach integrates external knowledge, such as the physical structure of the graph. AutoSTG [27] introduced the concept of meta-learning. A meta-learner first analyzes the structure of a physical graph, such as a road network, to generate customized operations, not using raw distance graphs. AutoSTG+ [28] advanced this concept by utilizing multiple meta-graphs. These additional graphs represent physical connectivity and other relationships, such as connections between nodes with similar traffic patterns. The multiple meta-graphs allow the model to capture more complex and latent spatio-temporal dependencies.

While these NAS-based approaches have outperformed manually designed models, they share a common limitation. Most studies focus almost exclusively on maximizing predictive accuracy but leave model efficiency as a secondary concern (i.e., parameter count, computational cost (FLOPs), and inference speed). The research on SimpleSTG [29] analyzed the search space through grid searching to identify a high-performing set of architectures. However, its criteria for high-performing regions is only for predictive performance. Consequently, models discovered by conventional NAS-based STGNN often require extensive training time to achieve high accuracy.

Unlike these methods, SearchLight introduces a multi-objective loss on NAS for STGNN that targets both lightweight and accurate architectures. Also, while prior NAS-based works use all candidate operations for each cell, SearchLight sets cell types and divides the operation set into spatial and temporal cells. Therefore, SearchLight can explicitly handle spatial and temporal data and reduce the search space if candidate operations increase.

III. BACKGROUNDS

This section provides the background information for understanding our work, Spatio-Temporal Graph Neural Networks (STGNNs) and Neural Architecture Search (NAS). Section III-A describes the key components and principles of STGNN, and Section III-B outlines the general concept and approaches of NAS.

A. SPATIO-TEMPORAL GRAPH NEURAL NETWORK

Spatio-Temporal Graph Neural Network (STGNN) is a neural network for various spatio-temporal tasks [1], [2], [3], [4], [6], [11], [12], [15], [32], [34], [35], [36], [37], [38] powered by Graph Neural Networks (GNN) [39] used for various domains such as traffic networks, epidemics, weather forecasting, object detection, motion capturing, or water management.

The inner structure of the STGNN model varies, but it usually handles spatial and temporal information separately and combines both processed information in the model. The spatial block contains GNN-based layers, and the temporal block contains time series layers such as convolution attention or recurrent networks. The model can have multiple blocks, and the order of stacked blocks is different in various studies. We introduce the preprocessing stage of STGNN, which appears in most of the STGNN models. Figure 1 provides a visual example of the raw data processing workflow before training an STGNN model. The raw data, which includes sensor locations on a map and time-series signals from sensors, is processed through two separate pipelines: spatial and temporal information.

First, a distance graph A is created depending on the sensor locations. The Gaussian kernel maps each distance $A_{u,v}$ between the vertex u and v to $A'_{u,v} \in [1, 0]$, which converts short distance to one and long distance to zero. Then, some value of A' under the threshold k, which represents a very far distance, is set to zero, resulting in a normalized adjacency matrix $\tilde{A}$ between 0 and 1. This process is written as formula

$$\tilde{A}_{u,v} := \begin{cases} A'_{u,v} & \text{if } A'_{u,v} > k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $A'_{u,v} = e^{-(A_{u,v}/\text{stdev}(A))^2}$, and $\tilde{A}_{v,v} = 1$. Some STGNNs use one of the Laplacian matrices instead of the normalized adjacency matrix $\tilde{A}$, depending on the model design. There are several types of Laplacian matrix, but the symmetrically normalized Laplacian is the most basic version written in Figure 1. In Laplacian matrix of Figure 1, we note that I is the identity matrix, and $D^{-\frac{1}{2}}$ is calculated by Equation (2). For any vertices $u, v \in \mathcal{V}$, $D^{-\frac{1}{2}}_{u,v}$ is the reciprocal of the square root of deg(i) when source vertex u and destination vertex v are the same; otherwise it is zero:

$$D^{-\frac{1}{2}}_{u,v} := \begin{cases} \dfrac{1}{\sqrt{\deg(v)}} & u = v \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\mathcal{V}$ is the set of vertices and $\deg(v) = \sum_{v \in \mathcal{V}} \tilde{A}_{u,v}$.

The sensor value matrix $X \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{V}|}$ is formed by presenting time series as a matrix with rows (timestamps) and columns (sensors), where $\mathcal{T}$ is the set of timestamps. The dataset is split into the training set, validation, and testing datasets based on a predefined ratio, with its timestamp $\mathcal{T}_{\text{train}}$, $\mathcal{T}_{\text{val}}$, and $\mathcal{T}_{\text{test}}$, respectively. Since most STGNN models require fixed-size input lengths, the datasets are divided into slices using a sliding window [40]. This method makes overlapping windows of length $t_\text{i} + t_\text{o}$ over all timestamps t in the given set (i.e., training, validation, testing set), generating slices $X_{t-t_\text{i}:t+t_\text{o},:} \in \mathbb{R}^{(t_\text{i}+t_\text{o}) \times |\mathcal{V}|}$ (for the notation, see Section Appendix).

The STGNN model tries to minimize loss $\mathcal{L}$ repeating multiple epochs. In each epoch, for every slice $X_{t-t_\text{i}:t+t_\text{o},:}$, the STGNN model M predicts $\hat{X}_{t:t+t_\text{o},:}$ using $X_{t-t_\text{i}:t}$ and Laplacian matrix L as

$$\hat{X}_{t:t+t_\text{o},:} = M(L, X_{t-t_\text{i}:t,:}). \tag{3}$$

The loss $\mathcal{L}$ returns the mean value of the difference between the predicted value $\hat{X}_{t:t+t_\text{o},:}$ and the actual value $X_{t:t+t_\text{o},:}$. The optimizer finally updates model M's parameter θ that can minimize $\mathcal{L}$ as much as possible.

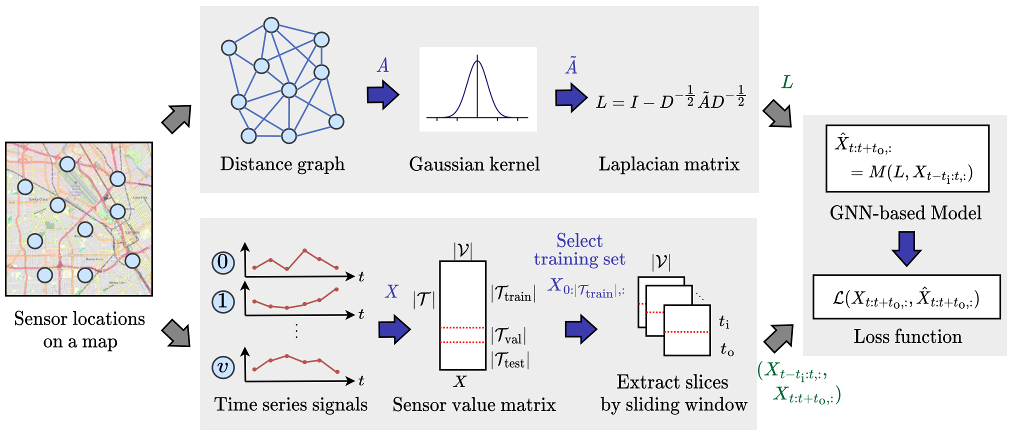

Sensor locations on a map
Distance graph
A
Gaussian kernel
$\tilde{A}$
$L = I - D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}$
Laplacian matrix
L
$\hat{X}_{t:t+t_o,:} = M(L, X_{t-t_i:t,:})$
GNN-based Model
$\mathcal{L}(X_{t:t+t_o,:}, \hat{X}_{t:t+t_o,:})$
Loss function
0
1
v
t
Time series signals
X
$|\mathcal{V}|$
$|\mathcal{T}|$
$|\mathcal{T}_{\text{train}}|$
$|\mathcal{T}_{\text{val}}|$
$|\mathcal{T}_{\text{test}}|$
Sensor value matrix
Select training set
$X_{0:|\mathcal{T}_{\text{train}}|,:}$
t_i
t_o
Extract slices by sliding window
$(X_{t-t_i:t,:}, X_{t:t+t_o,:})$

FIGURE 1. Preprocessing pipeline of spatio-temporal data for the STGNN model training.

B. NEURAL ARCHITECTURE SEARCH

Neural Architecture Search (NAS) is an automated technique for discovering effective neural network structures. It explores design choices such as the types of layers, the placement of skip connections, and the selection of appropriate hyperparameters. Traditionally, the design of neural network architectures has relied heavily on the expertise and intuition of researchers. This dependency makes it challenging for non-experts or beginners to adapt existing architectures to their specific tasks. Moreover, human-designed architectures may be constrained by prior knowledge and fixed design patterns, potentially limiting the discovery of novel and more effective architectures. NAS aims to reduce manual effort by automatically enabling algorithms to generate neural architectures with minimal human effort. Many approaches to the architecture search have been studied, such as NAS-RL [41], MetaQNN [42], Large-scale Evolution [43], NASNet [44], PNAS [45], ProxylessNAS [46], AmoebaNet [47] and DARTS [30].

One of the earliest works in NAS is NAS-RL [41], which introduces a reinforcement learning (RL)-based approach to exploring neural network architectures. The method applies the agent–environment paradigm from RL, where the controller functions as the agent, and the generated neural architecture serves as the environment. The controller is implemented as a recurrent neural network (RNN) that generates sequences of tokens, each representing a design choice, such as the number of filters, filter height, width, or stride dimensions in a convolution layer. A candidate architecture is constructed and trained from scratch based on the generated sequence to evaluate its classification accuracy. This accuracy score is used as the reward signal in the RL framework. If the architecture yields poor performance, the controller receives a low reward and is encouraged to sample a different architecture in the next iteration. Through repeated trials, the controller gradually learns to generate architectures that lead to higher performance. While conceptually straightforward, NAS-RL suffers from major efficiency issues. The search process explores enormous architecture design space, and every architecture must be trained from the beginning to obtain its performance, which is computationally expensive. According to the original paper, it can take several thousands of GPU days to search for an architecture on the CIFAR-10 dataset [48] using a single GPU, making the method impractical for large-scale or time-sensitive applications.

NASNet [44] is affected by hints that modern deep neural networks often consist of repeated structures, such as stacked layers with skip connections, as seen in ResNet [49]. Instead of searching the whole network structure, NASNet defines a smaller unit called a cell (a repeatable block of layers) and focuses on searching only the cell design. Unlike NAS-RL, which outputs the entire architecture, NASNet uses a controller RNN to generate a cell structure described as a sequence of operations and connections and then constructs the whole network by stacking copies of this cell. This approach significantly reduces the search complexity and computation, reducing 10 times the searching time compared to NAS-RL (approximately 2,000 GPU days on CIFAR-10). PNAS [45] further improves efficiency by progressively constructing and deciding on the inner cell structure, which can reduce the search space more. Despite being about five times faster than NASNet (225 GPU days on CIFAR-10), the method still requires enormous computational resources, making it challenging to apply in real-world settings.

A limitation of earlier NAS methods is that each candidate architecture must be trained from scratch, making the overall search process extremely time-consuming. DARTS (Differentiable Architecture Search) [30] addresses this issue using a more efficient gradient-based approach. Instead of selecting and training a single operation per layer at a time, DARTS combines all possible operations as a weighted sum and alternately updates the operation's parameter and its weight. This approach saves much time because it allows all candidate operations to be evaluated simultaneously. DARTS identifies the most effective operation by selecting the one with the highest weight and reduces searching time significantly compared to earlier methods (0.25 to 4 GPU days on CIFAR 10).

IV. OUR METHOD: SEARCHLIGHT

This section explains our method, SearchLight, which aims to find lightweight Spatio-Temporal Graph Neural Network (STGNN) models empowered by Neural Architecture Search (NAS). We first present a general overview of SearchLight in Section IV-A. We then present the candidate operations and multi-objective loss that are the core components of SearchLight in Section IV-B and Section IV-C, respectively. The decision of candidate operations after the architecture search is described in Section IV-D. Finally, the selection of manual hyperparameters is explained in Section IV-E.

A. OVERVIEW

Figure 2 depicts the proposed model architecture and architecture search method. The model M consists of multiple cells C_k. Each cell sequentially transmits an outer hidden state h_{k+1}, which is an intermediate result. Each cell receives two inputs: h_k from the preceding cell and h_{k-1} from two steps prior. This structure acts as a skip-connection, which reduces information loss and stabilizes training. The model M has the form $\hat{X}_{t:t+t_o,:} = M(X_{t-t_i:t,:})$ and $h_0 := X_{t-t_i:t,:}$ (for the notation, see Section Appendix).

Definition 1 (Cell): A cell is a function mapping two outer hidden states to one hidden state written as

$$h_{k+1} = C_k(h_{k-1}, h_k), \tag{4}$$

where $k \geq 1$. Especially the first C_0 cell is $h_1 = C_0(h_0, h_0)$. Within each cell, there are multiple inner hidden states z_m. The mixed operation defined in Definition 2 between these states is denoted by $f_{i,m}$.

Definition 2 (Mixed Operation): A mixed operation $f_{i,m}$ is the weighted sum of each candidate operation g_j for mapping the cell inner hidden states $z_i \mapsto z_m$ $(i < m)$. Its weight is denoted as α such that

$$f_{i,m} = \sum_{j=0}^{m-1} \alpha_{i,j}^{(m)} g_j. \tag{5}$$

The weight α is represented as a matrix. If n is the number of candidate operations g_j, $\alpha^{(m)} \in \mathbb{R}^{m \times n}$. For any row index i, $\sum_{j=0}^{n-1} \alpha_{i,j}^{(m)} = 1$. The mixed operation exists during the architecture searching stage and is fixed to a specific operation for the training stage.

The state z_m is composed of the sum of the preceding states (i.e., $z_0, \ldots, z_{m-1}$) using mixed operation, which can be expressed as

$$z_m = \sum_{i=0}^{m-1} f_{i,m}(z_i), \tag{6}$$

where $2 \leq m < m_{\max}$. Here, $m_{\max}$ represents the number of inner hidden states in a cell. The cell's first and second inner hidden state are the same as the cell's outer state, defined as $z_0 := h_{k-1}, z_1 := h_k$.

Existing STGNN studies [29] often process spatiotemporal information separately before combining it. We use this approach by classifying cell types to **S-cell** (Spatial cell) and **T-cell** (Temporal cell), which are formally defined in Definition 3 and Definition 4, respectively.

Definition 3 (S-cell): S-cell is a cell that only uses a set of candidate operations containing a graph neural network with a Laplacian matrix L or an adjacency matrix A.

Definition 4 (T-cell): T-cell is a cell that only uses a set of operations specialized in time series data and does not utilize a graph neural network.

Note that if the cell type is the same, its internal structure, such as the number of inner hidden state $m_{\max}$ or the weight of candidate operation α, is the same. In contrast, the learnable parameter θ and the number of candidate operations n can be different. Therefore, the learnable parameter θ update order depends on the cell order. For example, when the cells are stacked in the order [S, T, S], the θ of the first S-cell and the third S-cell are separate objects. Since backpropagation occurs over the fully stacked cells (i.e., the model), the gradient is propagated from the closest cell to the output side to the input side.

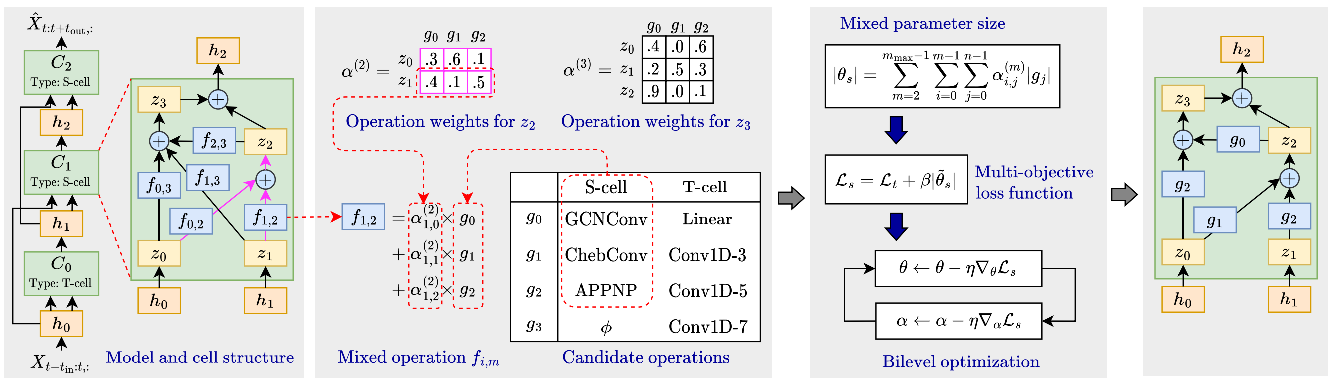
The cell type and order of the whole model are manually determined before starting the searching stage. It is represented by a string consisting of the S and T letters. For example, the cell order depicted in Figure 2(a) that C_0 is a T-cell, C_1 is an S-cell, and C_2 is an S-cell, can be abbreviated as TSS. The effect of different cell types and orders is further discussed in Section IV-E.

Once the cell type and order are determined, a loss function that considers both model size and prediction error is employed to identify a lightweight model. To achieve this, the operation weight α is used during the searching stage to calculate the model parameter size, which is incorporated into a multi-objective loss function. Then, the bilevel optimization process [30] converges the α values to identify the optimal cell structure. After sufficient convergence, the highest α values are selected to replace the mixed operation $f_{i,m}$ with the corresponding candidate operation g_j. After deciding on the architecture, the model conducts a typical neural network training and testing process.

B. CANDIDATE OPERATIONS

In this section, we explain candidate operations for S- and T-cells. S-cell is targeted to process spatial information and uses graph neural network operation. We select the following three graph operations as candidate operations for S-cells:

- **GCNConv**: Graph Convolutional Networks (GCNs) [50] propagate information along graph edges and aggregate it iteratively. GCN is one of the basic and widely used layers among GNN operations. For the input z_i, we denote the initial hidden state of GCN as $Y^{(0)} = z_i$. The GCN iterates the following formula over



(a) A model with two types of cells (b) Mixed operation and disjoint operation sets (c) Multi-objective architecture search (d) Discovered architecture

FIGURE 2. **Overview illustration of SearchLight.**

$0 \leq d \leq d_{\max}$,

$$Y^{(d+1)} = \sigma(\dot{A}Y^{(d)}\theta^{(d)}). \tag{7}$$

Note that $\dot{A} = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}$, $\tilde{A}$ is from Equation (1), and $\theta^{(d)}$ represents learnable parameters of the operation. The final output is $y_j = Y^{(d_{\max}+1)}$. To prevent excessive diffusion, we set $d_{\max} = 2$ to limit aggregating vertex information up to 3-hop neighbors.

- **ChebConv**: ChebNet [51] improves GCN adopting Chebyshev polynomials T_d to resolve several problems induced by the usage of the nested function of GCN. GCN may have the vanishing or exploding gradient problem and large size of learnable parameters because it uses nested functions to calculate aggregation on the graph. On the other hand, ChebConv approximates d-hop aggregation with a summation that has a lower calculation cost and minimizes the gradient problem without requiring deep layers. The output of ChebConv is computed as

$$y_j = \sum_{k=0}^{d_{\max}} T_d z_i \theta^{(d)}. \tag{8}$$

Here, $T_0 = I$, $T_1 = L$, and $T_d = 2LT_{d-1} - T_{d-2}$ recursively, where $L = I - D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}$. We set $d_{\max} = 3$ to aggregate information to 3-hop neighbors.

- **APPNP**: Approximate Personalized Propagation of Neural Predictions (APPNP) [52] is one of derived methods of GCN. APPNP only prepares learnable parameters at the initial input and propagates information iteratively using Personalized PageRank [53], [54]. Therefore, the fully connected layer computes initial hidden state $Y^{(0)} = \sigma(z_i\theta)$ for the layer input z_i, and the iterative propagation,

$$Y^{(d+1)} = (1-a)\dot{A}Y^{(d)} + aY^{(0)}, \tag{9}$$

is conducted until d reaches $d_{\max}$. Here, $\dot{A}$ is from Equation (7) and $0 \leq a \leq 1$. The final output is $y_j = Y^{(d_{\max}+1)}$, and we set $d_{\max} = 2$ to prevent propagation over 3-hop neighbors.

T-cell is expected to handle temporal information. We set the following two candidate operations for T-cells:

- **Linear**: A fully connected layer without hidden layers. This operation represents the simplest and most fundamental neural network computation. Additionally, we do not use an activation function. The output is computed as

$$y_j = z_i\theta + b, \tag{10}$$

where θ denotes the learnable parameter, and b represents the bias term.

- **Conv1D-κ**: We prepare a 1D convolution layer with kernel size κ, combined with adaptive average pooling, which fits the output of convolution size to final output y_j. The computation is expressed as

$$y_j = \text{AvgPool}(z_i * \theta), \tag{11}$$

where $*$ denotes the 1D convolution operation. We do not apply dilation or padding size to minimize design variation of the convolution. We set three distinct Conv1D for T-cell candidate operations with kernel size $\kappa \in \{3, 5, 7\}$.

C. MULTI-OBJECTIVE OPTIMIZATION

To search for a lightweight model with low prediction error, we use the multi-objective loss $\mathcal{L}_s$, combining a model prediction error and its parameter size. This design is inspired by several NAS studies that propose combined loss function strategies for multiple objectives [46], [55], [56], [57], [58], [59], [60].

Definition 5 (Multi-Objective Loss Function): A combined loss function, which is used only for the searching stage, is written as:

$$\mathcal{L}_s = \mathcal{L}_t + \beta|\tilde{\theta}_s|, \tag{12}$$

where $\mathcal{L}_t$ is a prediction error satisfying $\mathcal{L}_t \geq 0$, $|\tilde{\theta}_s|$ is a normalized parameter size (i.e., the number of total

parameters), and β is a scaling factor. the loss $\mathcal{L}_s$ is used exclusively for the searching stage, and after the model architecture is decided, $\mathcal{L}_t$ is used for the training stage.
The loss function $\mathcal{L}_s$ can be viewed as the Minimum Description Length (MDL) principle,

$$\ell_{\min}(M, D) = \ell(M) + \ell(D|M), \tag{13}$$

where D is data, M is model, and ℓ is length or size. MDL says that the best model minimizes the total length required to describe both the model and the encoded data. In this context, prediction error $\mathcal{L}_t$ is the code length of the data given the model $\ell(D|M)$, reflecting how efficiently the model compresses observed data. The number of model parameters $|\tilde{\theta}_s|$ corresponds to the model size $\ell(M)$. By minimizing multi-objective loss function $\mathcal{L}_s$, the discovered architecture is close to the lightweight-sized model, describing whole input data without overfitting.

Since $\mathcal{L}_t$ and $|\tilde{\theta}_s|$ in Equation (12) can be in different numerical ranges, a scaling factor β adjusts each contribution. This approach prevents the model from excessively prioritizing either prediction error or model size, and tries to equalize both terms' contributions. The value of β is determined during an initial warm-up phase in the searching stage, and does not change during the remaining searching stage. In detail, the β is decided by the mean of the prediction loss for the first few batches that

$$\beta = \frac{1}{|\mathcal{T}_w|} \sum_{\tau \in \mathcal{T}_w} \mathcal{L}_t(X_{\tau:\tau+t_o,:}, \hat{X}_{\tau:\tau+t_o,:}), \tag{14}$$

where $\mathcal{T}_w$ is the set of timestamps from the first few batches, and $\mathcal{T}_w$ is randomly selected among all timestamps for the searching stage. We set $|\mathcal{T}_w| = 10$ for this work.

β is expected to be similar to the mean of the prediction loss ($\beta \approx \overline{\mathcal{L}_t}$), because β is the sample mean of all batches' loss. Here, the overline notation on the top of each term is *the overall mean of the selected terms* during the searching stage. By this approximation, $\overline{\mathcal{L}_s} = \overline{\mathcal{L}_t} + \beta\overline{|\tilde{\theta}_s|} \approx \overline{\mathcal{L}_t}(1 + \overline{|\tilde{\theta}_s|})$ is satisfied where $0 \leq \overline{|\tilde{\theta}_s|} \leq 1$ because $0 \leq |\tilde{\theta}_s| \leq 1$ is guaranteed by Equation (15). Therefore, $\overline{\mathcal{L}_t} \leq \overline{\mathcal{L}_s} \lessapprox 2\overline{\mathcal{L}_t}$, where $\lessapprox$ notation is *approximately greater than*, and β can constrain the effect of the normalized parameter size $|\tilde{\theta}_s|$ to a similar value to $\mathcal{L}_s$.

$|\tilde{\theta}_s|$ in Equation (12) is the min-max scaled value of $|\theta_s|$, expressed as

$$|\tilde{\theta}_s| = \frac{|\theta_s| - \min|\theta_s|}{\max|\theta_s| - \min|\theta_s|}. \tag{15}$$

Here, $|\theta_s|$ is the mixed parameter size. Unlike the training stage, in which the model architecture is fixed, all candidate operations overlap during the searching stage, which makes it difficult to estimate the expected model size. The simple sum of the parameter sizes of all candidate operations does not reflect the changing model structure because it returns a constant model size over the searching process. We introduce the mixed parameter size in Definition 6 to address this issue. This idea is based on the observation that the architecture

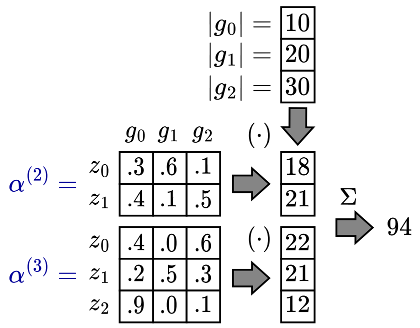


FIGURE 3. Example of calculating mixed parameter size.

weights α are continuously updated during the search. Using the α values as weights for each size of candidate operation, we can estimate the expected parameter size of the model at each point during the searching stage. This weighted sum allows us to track how the model size evolves during the searching stage.

Definition 6 (Mixed Parameter Size): Mixed parameter size $|\theta_s|$ is calculated as the sum of the products of operation parameter sizes $|g_j|$ and their corresponding weights $\alpha_{i,j}^{(m)}$, as defined in

$$|\theta_s| = \sum_{m=2}^{m_{\max}-1} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \alpha_{i,j}^{(m)} |g_j|. \tag{16}$$

The intuition for setting α for calculating $|\theta_s|$ is that it represents the relative contribution of each candidate operation during the searching stage. α can be interpreted as a probability of which candidate operations will be selected; thus, the weighted sum is considered an expectation of the parameter size. This approach can capture the dynamic change of model architecture throughout the searching stage.

Figure 3 illustrates a matrix-view example explaining the mixed parameter size in Equation (16), because it is the same as the inner product of matrices. We set two matrices for containing weight $\alpha_{i,j}^{(m)}$, and one matrix with parameter sizes $|g_j|$. We denoted z_i and g_j around matrices to distinguish which element corresponds to each inner hidden state z_i and candidate operation g_j. So, for example, each row of the weight matrix $\alpha^{(2)}$ represents candidate operation weights for mixed operations $f_{0,2}$ and $f_{1,2}$, that map $z_0 \mapsto z_2$ and $z_1 \mapsto z_2$, respectively. The $\alpha^{(2)}$ and $\alpha^{(3)}$ matrices take the inner product with the matrix of parameter size $|g_j|$. Then, all the elements of the result matrix by the inner product are totaled to get the mixed operation size $|\theta_s|$.

$\min|\theta_s|$ and $\max|\theta_s|$ in Equation (15) are defined as follows. Among all candidate operations g_j, the smallest parameter size $\min|g_j|$ and the largest parameter size $\max|g_j|$ are identified. For each, the α values in the j-th column are set to 1, and Equation (16) is computed. This yields

$$\min|\theta_s| = \sum_{m=2}^{m_{\max}-1} m \min|g_j|, \quad \max|\theta_s| = \sum_{m=2}^{m_{\max}-1} m \max|g_j|. \tag{17}$$

After setting the loss function, the bilevel optimization [30] is conducted to find the best suitable operation. Each candidate operation has parameters θ, and the operation's weight α is closely related to each other. If θ is not decided, all the operations g_j return meaningless random value, so we cannot decide which operation gets more weight α. On the other hand, if α is not decided, it is the same that all candidate operations g_j receive the same contribution ratio, which cannot be chosen in a single operation. Therefore, θ and α should be optimized together.

Algorithm 1 Bilevel Optimization of SearchLight

```
Data: $M_{\theta,\alpha}$, /* a model with mixed operations */
      $p$, /* timepoint in training set,
             satisfies $t_i \le p \le |\mathcal{T}_{\text{train}}| - t_o$ */
      $q$, /* timepoint in validation set,
             satisfies $|\mathcal{T}_{\text{train}}| + t_i \le q \le |\mathcal{T}_{\text{train}}| + |\mathcal{T}_{\text{val}}| - t_o$ */
      $\eta_s$, /* maximum number of epochs
              in searching stage */
      $\xi$, /* learning rate of searching stage */
      $\mathcal{L}_s$, /* loss function for searching stage */
1: for each epoch $< \eta_s$ do
2:     for each $p$ and $q$ do
3:         $\hat{X}_{q:q+t_o,:} \leftarrow M_{\theta,\alpha}(X_{q-t_i:q,:})$ /* use validation set */
4:         $\alpha \leftarrow \alpha - \xi \nabla_\alpha \mathcal{L}_s(X_{q:q+t_o,:}, \hat{X}_{q:q+t_o,:})$ /* update $\alpha$ */
5:         $\hat{X}_{p:p+t_o,:} \leftarrow M_{\theta,\alpha}(X_{p-t_i:p,:})$ /* use training set */
6:         $\theta \leftarrow \theta - \xi \nabla_\theta \mathcal{L}_s(X_{p:p+t_o,:}, \hat{X}_{p:p+t_o,:})$ /* update $\theta$ */
7:     end for
8: end for
9: return $\alpha$
```

Algorithm 1 explains the bilevel optimization algorithm. The process iterates until a specified maximum epoch limit is reached (Lines 1–8 in Algorithm 1). For each timepoint p of the training set and timepoint q of the validation set, α and θ are alternately updated (Lines 2–7 in Algorithm 1). The process starts with the prediction using a validation batch (Line 3 in Algorithm 1). The loss $\mathcal{L}_s$ is computed, and α is updated via the gradient descent step to approach the optimal value (Line 4 in Algorithm 1). Using the updated α, the model $M_{\theta,\alpha}$ generates predictions on a training batch (Line 5 in Algorithm 1). The loss of the predictions is calculated, and θ is updated through gradient descent (Line 6 in Algorithm 1). This alternating update process converges α and θ toward optimal values. The final α values are returned for the model architecture selection process (Line 9 in Algorithm 1).

D. OPERATION DECISION

The operation decision process determines the model architecture by deciding candidate operations g_j to replace each mixed operation f_i, m for every cell type. In this step, we choose the two most influential candidate operations after the bilevel optimization of Algorithm 1, by selecting the operations corresponding to the two most significant α matrix values.

Algorithm 2 outlines the operation decision process. For each $2 \le m < m_{\max}$, the procedure in Lines 2–12 of Algorithm 2 is repeated. An empty list $\mathcal{R}$ is initialized for each m to store temporary results (Line 2 in Algorithm 2). For each i-th row of the $\alpha^{(m)}$ matrix, the maximum value and its corresponding index are identified and stored in list $\mathcal{R}$ (Lines 3–6 in Algorithm 2). The two largest $\alpha^{(m)}_{i,j^*}$ values (top-2) are selected, and their row indices i are stored in $\mathcal{I}$ (Lines 7–9 in Algorithm 2). The tuples in $\mathcal{R}$ are iterated to fix the mixed operation into a single operation (Lines 10–12 in Algorithm 2). If the row index i is in the top-2 row index list $\mathcal{I}$, the mixed operation $f_{i,m}$ is replaced with the candidate operation g_{j^*}; otherwise, the mixed operation which maps $z_i \mapsto z_m$ is eliminated (Line 11 in Algorithm 2).

Algorithm 2 Operation Decision of SearchLight

```
Data: $\alpha^{(m)}$, /* operation weight matrix
                 corresponding to cell inner state $z_m$ */
      $f_{i,m}$, /* mixed operation for $z_i \mapsto z_m$ */
1: for each $m$ do
2:     $\mathcal{R} = [\ ]$
3:     for each $i$-th row do
4:         Find max value $\alpha^{(m)}_{i,j^*}$ and its coordinate $(i, j^*)$
5:         $\mathcal{R}$.append($(i, j^*, \alpha^{(m)}_{i,j^*})$)
6:     end for
7:     Sort $\mathcal{R}$ by $\alpha^{(m)}_{i,j^*}$ in descending order
8:     $\mathcal{R}' \leftarrow \mathcal{R}[:2]$ /* select top-2 elements */
9:     $\mathcal{I} := \{\tau[0] \mid \tau \in \mathcal{R}'\}$ /* $\tau$ means tuple */
10:    for each $(i, j^*, \_) \in \mathcal{R}$ do
           /* _ is a placeholder */
11:        if $i \in \mathcal{I}$ then set $f_{i,m} := g_{j^*}$ else eliminate $f_{i,m}$
12:    end for
13: end for
```

Figure 4 provides an example of Algorithm 2. The example shows the same two processes for $\alpha^{(2)}$ and $\alpha^{(3)}$, and we explain the operation decision process in the case of $\alpha^{(3)}$. For each row $i \in \{0, 1, 2\}$, we can find the maximum value of each row of $\alpha^{(3)}$, and memorize its row index i and column index j^*, that $\{(0, 2), (1, 1), (2, 0)\}$. Among the maximum value of each row, the top-2 value is $\{0.6, 0.9\}$ which corresponds to the matrix coordinate $(i, j^*) \in \{(0, 2), (2, 0)\}$. Therefore, mixed operations are set to $f_{i,3} = g_{j^*}$ that $f_{0,3} = g_2$ and $f_{2,3} = g_0$, while $f_{1,3}$ is eliminated.

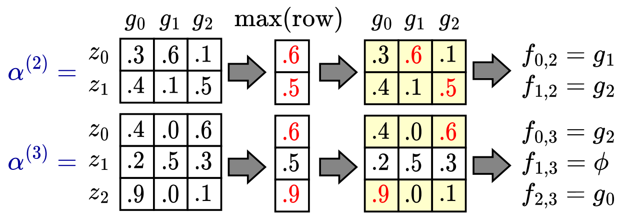
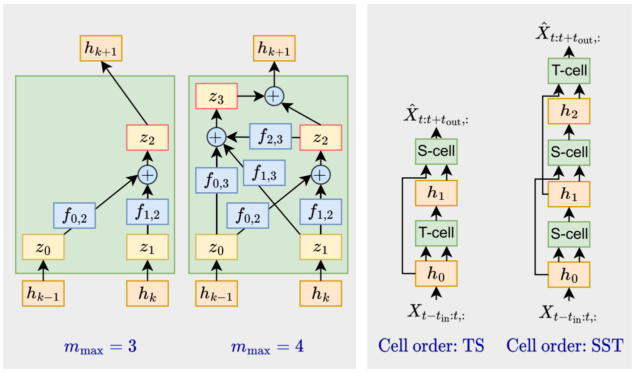


FIGURE 4. Example of the operation decision process by $\alpha^{(m)}$.



(a) The number of inner hidden state (b) The number and order of cell

FIGURE 5. Difference in cell and model structures depending on manual hyperparameters.

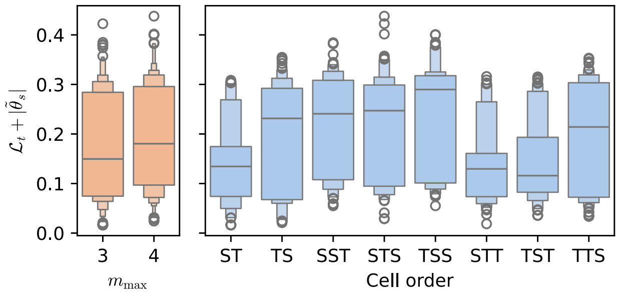


FIGURE 6. Pilot test result for selecting the high-performing hyperparameters before the evaluation, showing model performance distribution as box plots.

E. HYPERPARAMETER SELECTION

Although NAS automatically finds the architecture, some hyperparameters should be manually specified. Figure 5 shows manual hyperparameters and their impact on model structure. Figure 5(a) illustrates how the number of inner hidden states $m_{\max}$ affects the structural difference of cells. For $m_{\max} = 4$, an additional inner hidden state z_3 and corresponding mixed operations $f_{0,3}$, $f_{1,3}$, and $f_{2,3}$ are introduced compared to the case of $m_{\max} = 3$. Figure 5(b) shows the architecture resulting from different permutations of S-cells and T-cells. Cell order is represented as a string of S and T, sorted by the closest cell type to the model's input. The cell order determines the number of outer hidden states h_k and corresponding skip connections.

We pre-tested the performance of each hyperparameter using only the training and validation set to find the appropriate hyperparameter before the evaluation of Section V. For the performance score for this test, we combine prediction error $\mathcal{L}_t$ and normalized model parameter size $|\tilde{\theta}_s|$ as shown in Equation (12). Because we use MAAPE [61] for $\mathcal{L}_s$, whose range is normalized in $0 \leq \mathcal{L}_t \leq 1$, and the range of the normalized model parameter is $0 \leq |\tilde{\theta}_s| \leq 1$ by Equation (15). Because the numerical range of $\mathcal{L}_t$ and $|\tilde{\theta}_s|$ is equal, we set the scaling factor $\beta = 1$ from Equation (12), denoted as $\mathcal{L}_t + |\tilde{\theta}_s|$ in Figure 6.

Figure 6 shows testing results. As shown in the left plot of Figure 6 that compares $m_{\max}$, a slight difference in average score exists between the two values. For lightweight and efficient models, we select $m_{\max} = 3$ for the evaluation. Several cell orders are also evaluated in the right plot of Figure 6, containing at least one S-cell and one T-cell. Based on configurations yielding high performance (low mean error) and stable results (lower variance), the cell orders ST, STT, and TST are chosen.

TABLE 2. Specification of datasets.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{T}\rvert$	Start	End	Period
METR-LA [1]	207	34272	2012-03-01	2012-06-27	5 min.
PEMS-BAY [1]	325	52116	2017-01-01	2017-06-30	5 min.
PEMS03 [34]	358	26208	2018-09-01	2018-11-30	5 min.
PEMS04 [34]	307	16992	2018-01-01	2018-02-28	5 min.
PEMS07 [34]	883	28224	2017-05-01	2017-08-06	5 min.
PEMS08 [34]	170	17856	2016-07-01	2016-08-31	5 min.
PEMSD7 [3]	228	12672	2012-05-01	2012-06-13	5 min.
Air-PM25 [62]	172	8664	2014-05-01	2015-04-30	1 hour
Opinet [63]	228	3653	2015-01-01	2024-12-31	1 day
NOAA-958 [64]	958	4748	2010-01-01	2022-12-31	1 day
Uzel2022 [65]	58	3312	-	-	1/3 s.

V. EVALUATION

This section covers a comprehensive evaluation of SearchLight. Section V-A presents the experimental setup, datasets used, and baseline models for comparison. Section V-B shows the excellence of SearchLight in various metrics compared to other experiments. Section V-C analyzes the correlation between the discovered model's size and prediction error. Section V-D examines the relationship between training time and prediction error for the learned model, and Section V-E evaluates the efficiency of prediction error relative to the time required for model search. Section V-F presents the structure of the discovered model, and Section V-G shows ablation study on the multi-objective loss function.

A. ENVIRONMENT

1) SETTINGS

All experiments are conducted on a GPU-equipped machine featuring two Intel Xeon E5-2630v4 processors (2.20 GHz, each with 10 cores and 20 threads), 512 GB of main memory, and eight NVIDIA GTX 1080Ti GPUs. Each experiment runs independently using a single GPU. The dataset is divided into training, validation, and testing sets with ratios of 0.7, 0.1, and 0.2, respectively. For methods employing NAS, the searching stage utilizes only the training and validation sets to explore models, as described in Algorithm 1. For other cases, the training process uses the training and validation sets, while the testing process evaluates performance on the testing set. We set 100 epochs for searching and training regardless of the comparison target. We measure prediction error as Mean Arctangent Absolute Percentage Error (MAAPE) [61].

2) DATASETS

We obtain datasets from various domains as described below. The specifications of these datasets are provided in Table 2.

- **Traffic**: Several STGNN researches have used highway traffic datasets such as **METR-LA**, **PEMS-BAY**, **PEMS03**, **PEMS04**, **PEMS07**, **PEMS08**, and **PEMSD7** in different locations and era. All datasets are from the Freeway Performance Measurement (PeMS) System of California Department of Transportation [66]. The dataset measures average car speed in 5-minute intervals.
- **Urban environment**: The Particulate Matter (PM) below 2.5 micrometers dust dataset [62] contains hourly measurements from several locations across China. We extract **Air-PM25** dataset about one year of measurements from 172 air monitoring stations around Beijing and Tianjin.
- **Consumer prices**: Korea National Oil Corporation collects daily gasoline prices and publishes the price on the website **Opinet** [63]. We extract raw data from this website over ten years and then aggregated daily averages for 228 regions in South Korea. We set the region's location by the latitude and longitude of each region's town hall.
- **Meteorology**: **NOAA-958** have global daily temperatures around the world. This dataset is provided by the National Centers for Environmental Information [64]. We choose 12 years of temperature data from 958 randomly selected weather stations.
- **Neuroscience**: A neurobiological research [65] captures neuronal activation of lab animal C. elegans for 30 minutes. We call this dataset as **Uzel2022**. They recorded every luminescence neuronal image at approximately three frames per second. They also measured the number of gap junctions and synapses between neurons. We set the number of connections between neurons as the value of each edge and the intensity of neurons as the value of each vertex.

3) COMPARISON METHODS

We divide the comparison methods used in evaluating the proposed method into three groups. The first group includes popular state-of-the-art (SOTA) STGNN models, **AGCRN**, **ASTGCN**, and **GWNet**, which are already introduced in Section II-A. The second group is lightweight STGNN models, designed to have lower spatial and temporal complexity than SOTA models.

- **BigST**: BigST [22] resolves the high computational complexity of STGNNs to the linear-complexity design. It extracts features from long sequences to compress past temporal information into compact embeddings and performs time series forecasting using a modified graph convolution with linear time complexity.
- **STLGRU**: Spatio-Temporal Lightweight Graph Gated Recurrent Unit (STLGRU) [23] employs a memory-augmented attention module and gating mechanism to capture extensive spatiotemporal correlations. The gating strategy retains critical information and discards irrelevant data, enabling efficient memory usage and training with fewer parameters.

The third group is the NAS approach for finding optimal STGNN models. Unlike the SOTA and lightweight groups, which present solid neural architecture, these methods conduct architecture searching stages and produce different architectures using input datasets. We perform model searches using codes from those studies' repositories and evaluate their performance on discovered models.

- **AutoCTS**: Automated Correlated Time Series Forecasting (AutoCTS) [24] employs a hierarchical search to optimize the internal structure of ST-blocks (micro-search) for spatiotemporal processing and determines the optimal topology for connecting ST-blocks to form the overall model (macro-search).
- **AutoSTG**: AutoSTG [27] leverages meta-learning to predefine weights for graph and temporal convolutions based on given graph topologies. This approach automatically searches and constructs spatiotemporal graph neural network architectures tailored to the unique characteristics of each dataset.

Finally, our method SearchLight is compared to these three groups, marked as **SearchLight-ST**, **SearchLight-STT**, and **SearchLight-TST**, which corresponds to the cell orders of ST, STT, and TST, respectively. This cell order is found in Section IV-E.

B. OVERALL RESULTS

Figure 7 compares the average performance of SearchLight against other methods across four key metrics: model parameter size, training time, prediction error, and architecture searching time. Each annotation of bars indicates how many times smaller or faster SearchLight is (↓) or the percentage point difference (%p) in prediction error compared to other methods. Overall, SearchLight achieves significant efficiency benefits in terms of time and space while maintaining prediction performance.

The first and second subplots illustrate efficiency in model parameter size (KB) and training time per epoch (s.). Across all datasets, SearchLight (blue bars) consistently achieves smaller model sizes and shorter training times compared to baseline methods (gray bars). Model parameter size is reduced over a hundred times (e.g., $\times 121.3$ reduction on PEMS03), and training time is shortened similarly (e.g., $\times 119.0$ reduction on PEMS07). The mean reduction rate is $\times 103.0$ for model parameter size and $\times 74.0$ for training time. These results show that SearchLight is suitable for resource-constrained scenarios.

The third subplot presents the prediction error (%). SearchLight shows slightly larger prediction errors based on this subplot than the other methods, which is considered

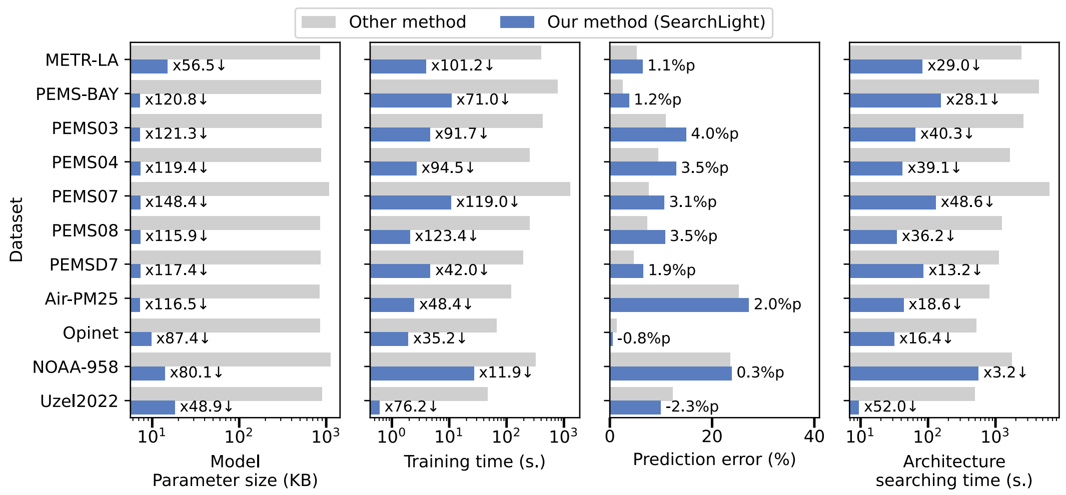


FIGURE 7. **Comparisons of the average performance across four metrics (i.e., model parameter size, training time, prediction error, and architecture searching time) between SearchLight and other baseline methods (i.e., mean performance of existing SOTA STGNN models, lightweight STGNN models, and NAS-based STGNN models). For the architecture searching time, only the mean of the NAS-based STGNN models is compared to SearchLight.**

a trade-off for reducing the model size. Nevertheless, the performance drop is relatively small compared to the reduction in the model size. For instance, on METR-LA, other methods achieve an average prediction error of 5.3%, whereas SearchLight shows only a 1.1%p error increase while reducing the model parameter size by 56.5 times. Similarly, on PEMS03, which exhibits the most significant performance drop, other methods yield an average error of 10.9%, while SearchLight sacrifices 4.0%p of accuracy but achieves a $\times$121.3 reduction in model size. Even with a 4.0%p performance drop, the resulting error remains 14.9%, corresponding to an 85.1% prediction accuracy. The mean of the performance drop is about 1.6%p.

We further count the number of datasets according to the performance difference. Two datasets (Opinet and Uzel2022) show lower prediction error than the baseline methods. Raising the criteria to 1%p difference includes one more dataset (NOAA-958), so the total number of datasets in the criteria is three. There are six datasets up to 2%p error increase (54.5% of the total datasets) by adding METR-LA, PEMS-BAY, and PEMSD7. For up to 3%p, it becomes seven datasets (63.6% of the total) by adding Air-PM25, and for up to 4%p, it becomes 10 datasets (90.9% of the total) by adding PEMS04, PEMS07, and PEMS08. In other words, SearchLight achieves up to a 3%p performance loss for nearly two-thirds of the datasets while reducing the model size by tens to hundreds of times.

The final subplot compares SearchLight's architecture searching time (s.) over other NAS for STGNN methods. SearchLight shows significantly shorter searching times such as $\times$52.0 reduction on Uzel2022) across most datasets. The mean searching time reduction is $\times$29.5. The result specifies that SearchLight identifies lightweight architectures rapidly. Overall, Figure 7 demonstrates that SearchLight efficiently finds a lightweight model that requires less space and time complexity to maintain prediction error.

C. ANALYSIS: MODEL SIZE AND ERROR

Figure 8 displays a scatter plot comparing model parameter size and prediction error across datasets. Each subplot represents results for a specific dataset, with the horizontal axis showing model parameter size on a logarithmic scale and the vertical axis indicating prediction error. Lower vertical axis values denote superior predictive performance. A gray dotted line illustrates the regression trend, which is computed as a linear regression in the log–linear space, thus corresponding to a logarithmic regression in the original scale. For methods using NAS, including the proposed SearchLight, all results are the performance of the discovered model. SearchLight models are marked with circular markers corresponding to the three candidate cell orders selected in Section IV-E.

In Figure 8, among the three models discovered by SearchLight, SearchLight-TST demonstrates the best performance, but shows a slight performance degradation for some datasets. We count each performance drop manually, and two datasets (Opinet and Uzel2022) show lower prediction error than the baseline models. Expanding the criteria to up to 1%p difference, four datasets are selected, including PEMS-BAY and NOAA-958. Up to 2%p, nine datasets (81.8% of total datasets) meet the criteria, including METR-LA, PEMS04, PEMS07, PEMS08, and PEMSD7. All datasets are within 3%p difference, including 2.2%p for PEMS03 and 2.0%p for Air-PM25. The mean of the performance drop of SearchLight-TST is about 0.7%p.

We can also examine the performance trade-off by examining individual datasets in Figure 8. Datasets below

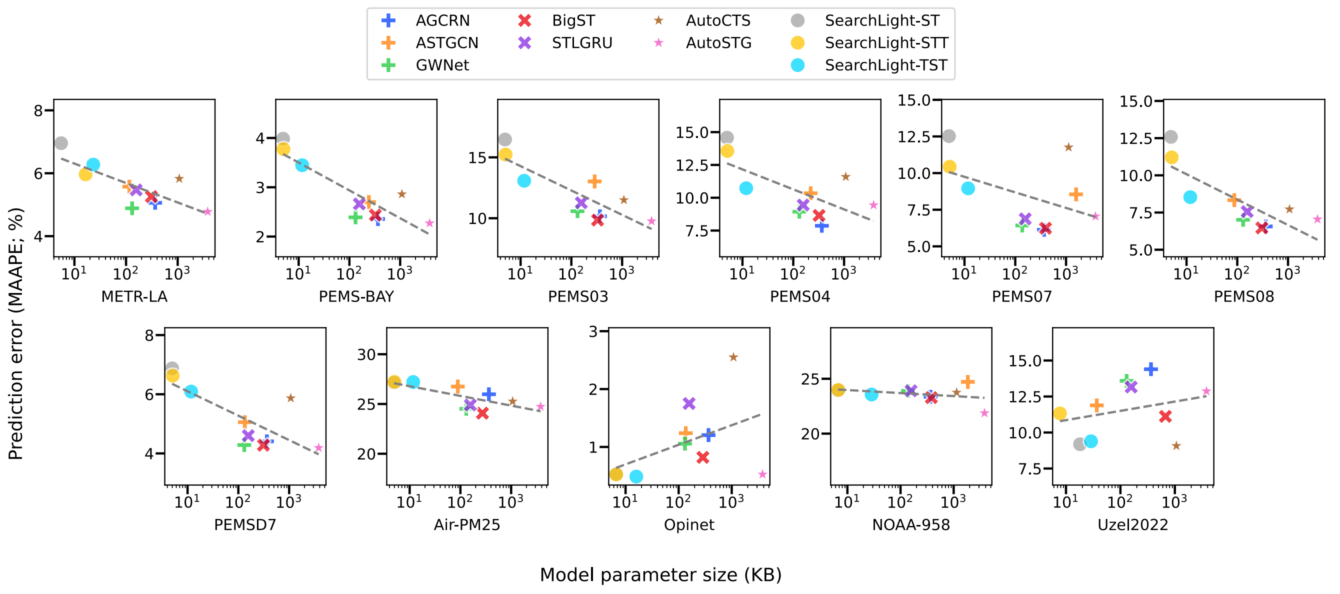


FIGURE 8. Relationship between parameter size and model performance. + is SOTA STGNN models, × is lightweight STGNN models, ★ is the discovered model by existing STGNN with NAS methods, and ● is the discovered model by our method. We organize all models in each column of the legend by their category. Each subplot corresponds to a different dataset. The horizontal axis shows parameter size on a logarithmic scale, while the vertical axis indicates prediction error on a linear scale, which can visually amplify differences along the y-axis rather than the x-axis. The gray dotted line represents a regression fitted under log–linear coordinates (i.e., equivalent to a logarithmic regression in the original scale).

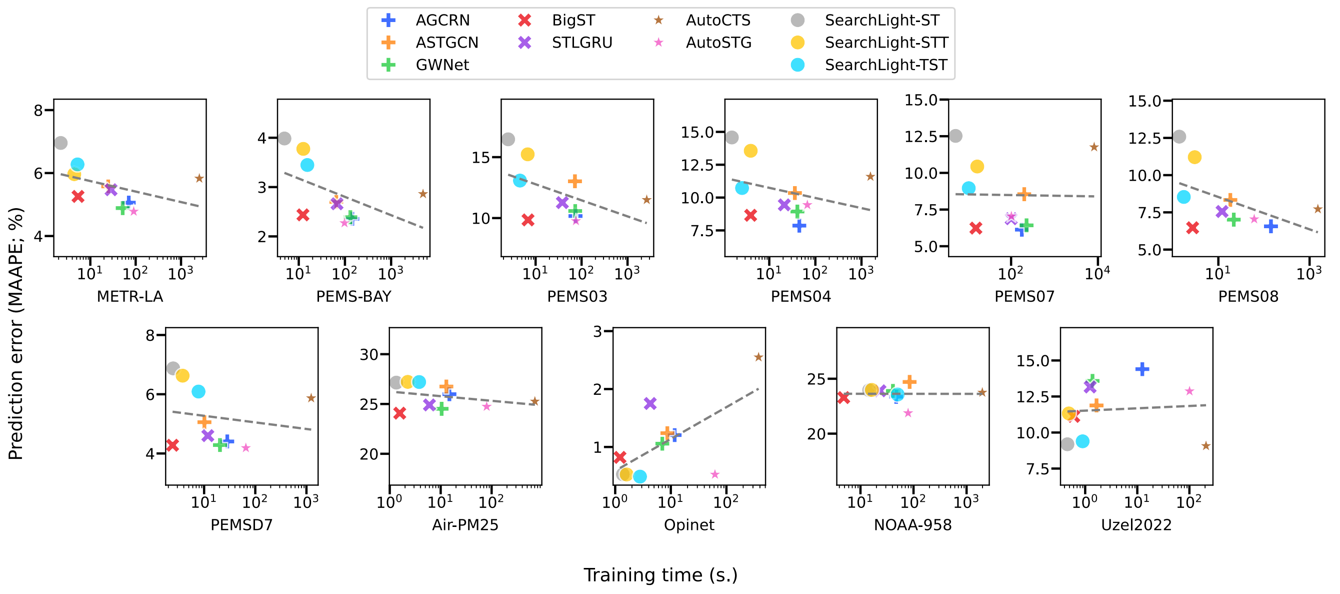


FIGURE 9. Relationship between per-epoch training time and model performance. + is SOTA STGNN models, × is lightweight STGNN models, ★ is the discovered model by existing STGNN with NAS methods, and ● is the discovered model by our method. The horizontal axis shows training time on a logarithmic scale, while the vertical axis indicates prediction error on a linear scale, which can visually amplify differences along the y-axis rather than the x-axis. The gray dotted line represents a regression fitted in log–linear space, showing the general tendency across models.

the regression line (dotted grey line in Figure 8) indicate better trade-offs, achieving lower prediction error at similar model sizes or smaller model sizes at similar error levels. For six datasets (PEMS03, PEMS04, PEMS07, PEMS08, Opinet, and Uzel2022), SearchLight-TST exists clearly below the regression line. It indicates that our method is closer to the trade-off limit of model size and prediction error than other models. In other words, our study shows lower error at the same-sized model or a smaller model size at the same error levels. The results confirm that SearchLight finds a lightweight structure for the given spatio-temporal data while balancing prediction performance. It implies that SearchLight

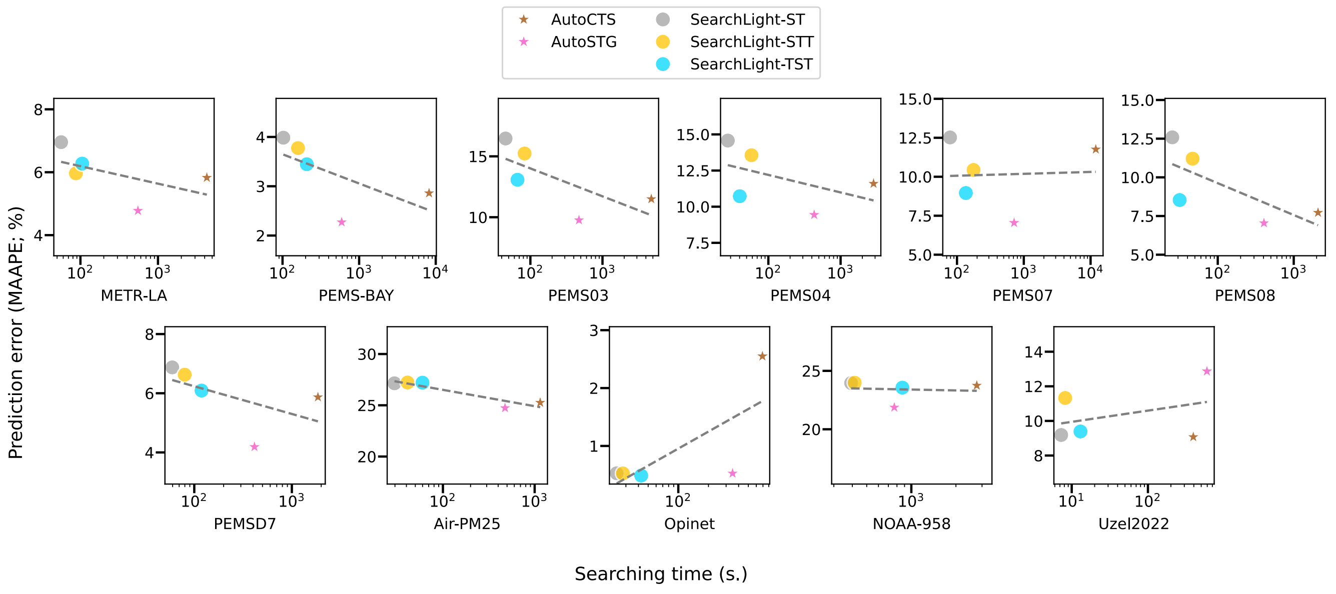


FIGURE 10. Relationship between per-epoch searching time and model performance. ★ is the discovered model by existing STGNN with NAS methods, and ● is the discovered model by our method (The SOTA STGNN and lightweight STGNN models appeared in Figure 8 and Figure 9 are not included because they are designed manually and not searched by NAS). The horizontal axis shows searching time on a logarithmic scale, while the vertical axis indicates prediction error on a linear scale, which can visually amplify differences along the y-axis rather than the x-axis. The gray dotted line represents the regression trend fitted under log–linear coordinates.

can be a robust and practical solution for resource-constrained systems.

D. ANALYSIS: TRAINING TIME AND ERROR

Figure 9 visualizes the relationship between the average per-epoch training time and the prediction error for each model across various spatio-temporal datasets. The horizontal axis represents training time on a logarithmic scale, and the vertical axis denotes the prediction error. Similar to Figure 8, the dotted line indicates a regression computed in log–linear space, capturing a logarithmic tendency in the original scale. SearchLight models are denoted by circular markers with the manual hyperparameter settings shown in Section IV-E.

Overall, the time-error results of Figure 9 are more sophisticated than the previous size-error results in Figure 8. For some traffic datasets, such as PEMS04 and PEMS08, our method is under the regression line, which achieves a better trade-off between training time and prediction error. However, for most other traffic datasets, our results are located above the regression line with remaining headroom for optimization in terms of time complexity. These results can come from two reasons. First, the current objective function $\mathcal{L}_s$ of SearchLight includes parameter size, not training time. Since the model is primarily optimized for parameter size, shorter training time is the additional expected effect from smaller model sizes. Second, we do not design SearchLight to optimize the time complexity of each candidate operation layer by layer. Figure 9 shows that BigST model performs better in some datasets. This is likely because BigST modifies each graph convolution to target linear

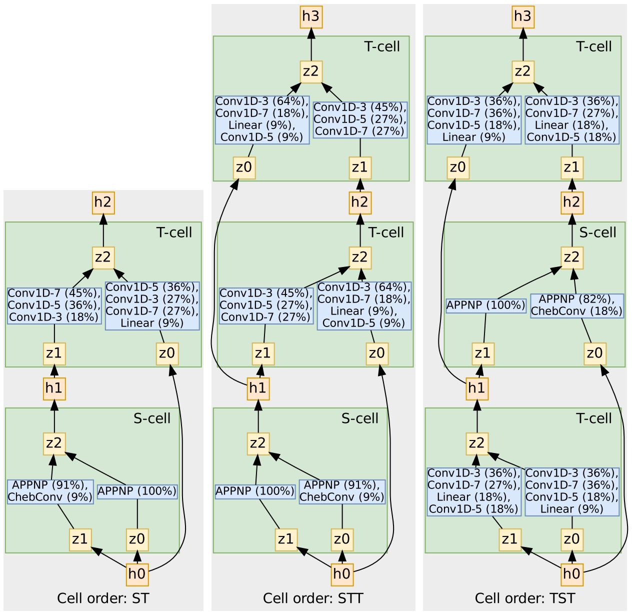

h3
T-cell
z2
Conv1D-3 (64%), Conv1D-7 (18%), Linear (9%), Conv1D-5 (9%)
Conv1D-3 (45%), Conv1D-5 (27%), Conv1D-7 (27%)
z0
z1
h2
T-cell
Conv1D-7 (45%), Conv1D-5 (36%), Conv1D-3 (18%)
Conv1D-5 (36%), Conv1D-3 (27%), Conv1D-7 (27%), Linear (9%)
h1
S-cell
APPNP (91%), ChebConv (9%)
APPNP (100%)
h0
Cell order: ST
Cell order: STT
Cell order: TST
Conv1D-3 (36%), Conv1D-7 (36%), Conv1D-5 (18%), Linear (9%)
Conv1D-3 (36%), Conv1D-7 (27%), Linear (18%), Conv1D-5 (18%)
APPNP (100%)
APPNP (82%), ChebConv (18%)
Conv1D-3 (36%), Conv1D-7 (27%), Linear (18%), Conv1D-5 (18%)
Conv1D-3 (36%), Conv1D-7 (36%), Conv1D-5 (18%), Linear (9%)

FIGURE 11. The total layout of discovered models and the frequency of used operations by cell orders.

complexity for scalability. It means that different metrics can be improved depending on which model optimization strategy is chosen and which metric is prioritized.

Nevertheless, SearchLight generally exhibits significantly shorter training times except in BigST case for most of the dataset. In Air-PM25 and NOAA-958 datasets, our study can reduce training time while maintaining prediction error. Also, we demonstrate near-optimal performance for the Opinet

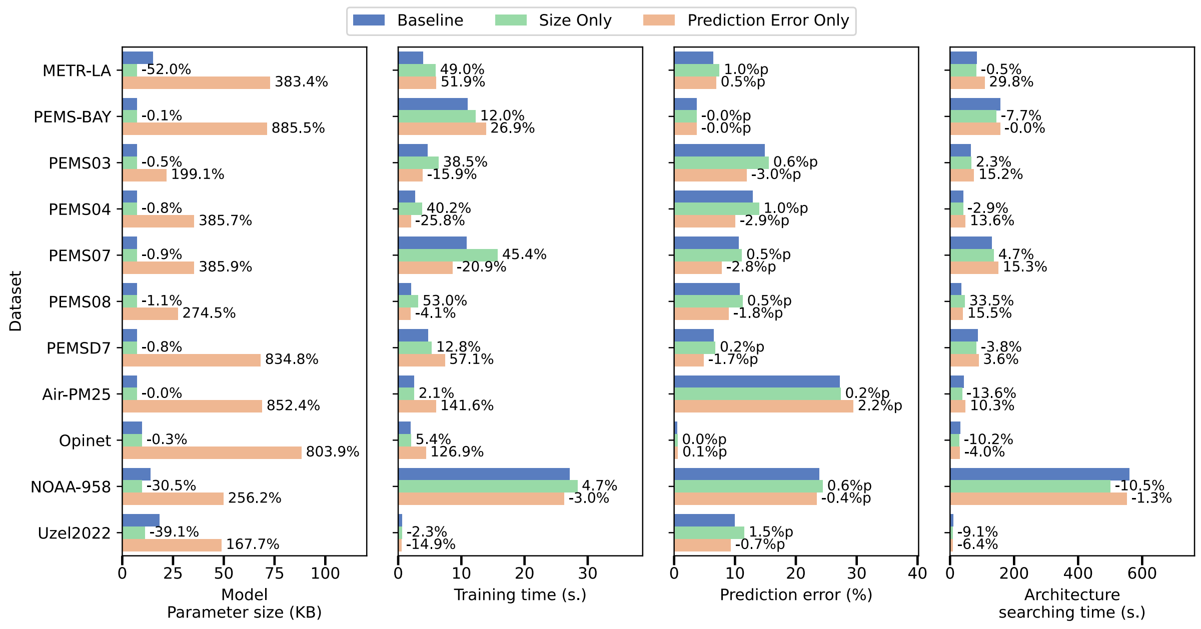


FIGURE 12. **Ablation study of the multi-objective loss function of SearchLight, across four metrics (i.e., model parameter size, training time, prediction error, and architecture searching time). *Size Only* refers to the case where only the model parameter size term remains in the loss function, while *Prediction Error Only* refers to the case where only the prediction error term remains. The numbers next to the bars indicate the relative increase or decrease compared to the baseline. For prediction error, the differences are reported in percentage points (%p), whereas the other metrics are reported as percentage (%) increases or decreases relative to the baseline.**

and Uzel2022 datasets that show the minimum trade-offs in training times and the lowest prediction errors. SearchLight has a remaining improvement margin for training time efficiency because the training time is not considered an objective function of architecture search. Nevertheless, our study can state that SearchLight can find lightweight models with less model size and shorter training time compared to most of the existing SOTA, lightweight, and NAS-based methods.

E. ANALYSIS: SEARCHING TIME AND ERROR

NAS methods iteratively apply bilevel optimization presented in Algorithm 1 to find optimal architectures with repeating epochs. It allows measurement of per-epoch searching time. Figure 10 visualizes the searching time versus prediction error for NAS-based STGNN models and our method. The horizontal axis represents searching time on a logarithmic scale, and the vertical axis is prediction error on a linear scale. A dotted regression line is added for clarity and fitted under log–linear coordinates (equivalent to logarithmic regression on the original scale).

SearchLight shows superior efficiency in searching time compared to similar NAS-based STGNN methods such as AutoCTS and AutoSTG. On datasets PEMS03, PEMS04, PEMS07, PEMS08, and Uzel2022, our results under the regression line indicate finding a more accurate model with less searching time. For Opinet dataset, SearchLight achieves the lowest prediction error with significantly reduced searching time, outperforming other NAS methods. Our study shows that a significantly better trade-off between searching time and model prediction error can be practically adopted in rapid model development and crucial updates.

F. LEARNED ARCHITECTURES

Figure 11 illustrates the overall model structures by SearchLight with cell orders ST, STT, and TST. We count and write the frequency of operations across all datasets. Blue boxes in each cell display the chosen operations and their selection frequency.

For S-cell, Figure 11 shows that APPNP is selected with over 80% frequency in all cases. This preference is because APPNP has the fewest learnable parameters that APPNP has only one fully connected layer. In contrast, other graph operations have multiple fully connected layers depending on the number of aggregation hops. For T-cell, 1D convolution operations are predominantly selected, likely due to fewer learnable parameters than Linear operations. The convolution kernel adaptively captures relevant temporal patterns by varying the kernel size from 3 to 7. As a result, SearchLight favors APPNP in S-cells for parameter efficiency and Conv1D with adjusted kernel sizes in T-cells. These choices demonstrate SearchLight's ability to achieve lightweight architectures while constructing data-adaptive model structures.

G. ABLATION STUDY

To validate the contributions of each term of Equation (12), we conduct an ablation study by eliminating each term.

TABLE 3. Mean changes of ablation study results across four metrics (i.e., model parameter size, training time, prediction error, and architecture searching time). All values represent the averaged relative changes or differences compared to the baseline, corresponding to the detailed results shown in Figure 12.

	Size only	Prediction Error Only
Model parameter size (change)	-11.5%	+493.5%
Training time (change)	+23.7%	+29.1%
Prediction error (difference)	+0.56%p	-0.94%p
Architecture searching time (change)	-1.61%	+8.34%

We set two cases, *Size Only* and *Prediction Error Only*. *Size Only* remains only size term of Equation (12) that becomes $\mathcal{L}_s = |\tilde{\theta}_s|$. The β term is unnecessary because there is only one term and no requirement to balance the effect of the performance and size. *Prediction Error Only* leaves only prediction error term which is $\mathcal{L}_s = \mathcal{L}_t$.

Figure 12 shows the performance changes by removing each term of the multi-objective loss function (Equation (12)). *Size Only* effectively reduces the number of parameters, but shows increased training time. As mentioned earlier in Section V-D, reducing model size does not necessarily improve training time. *Prediction Error Only* produces a huge model size because the algorithm chooses only to improve accuracy, not considering the parameter size, as shown in Section V-C. *Size Only* shows a slight increase in prediction error, while *Prediction Error Only* achieved lower error overall. Thus, optimizing only for accuracy improves performance but is very inefficient regarding model size, whereas optimizing only for size sacrifices accuracy. Finally, when examining the architecture searching time, both *Size Only* and *Prediction Error Only* show various time changes compared to the *Baseline*. Since one of the searching processes, bilevel optimization, includes training-like processes, slight differences in time can occur. It implies that search efficiency can vary depending on search space constraints and optimization objectives.

Table 3 reports the average results of the ablation study. For *Size Only*, the model parameter size decreases, but the prediction error is not directly minimized; the error increases by about 0.5%p. Training time increases by about 24%, but the exact cause is unclear. We analyze some of the learned architecture and we figure out the S-cell consistently selects GNN operations, but the T-cell chooses smaller kernel sizes for Conv1D-κ in *Size Only*. We presume that smaller kernels have narrower receptive fields and require more strides to cover the same time span, which may not utilize the GPU's bulk processing scheme and lead to increased training time. On the other hand, prediction error decreases by about 1%p for accuracy only, but the model parameter size increases nearly $\times 5$ compared to the *Baseline*, and training time also increased by about 30%. These results support that the multi-objective loss used in SearchLight effectively explores efficient network architectures by simultaneously considering both model size and accuracy.

VI. CONCLUSION

In this study, we propose SearchLight, a Neural Architecture Search (NAS) framework designed to automatically discover lightweight Spatio-Temporal Graph Neural Networks (STGNNs). Unlike existing approaches, SearchLight performs architecture search targeting the multi-objective goal of prediction accuracy and model size. To effectively capture the characteristics of spatio-temporal data, the model is constructed using two specialized cells, one for spatial features and the other for temporal features. Through benchmarking on real-world datasets, we show that SearchLight achieved an average of $\times 103.0$ reduction in model size and an average of $\times 74.0$ faster training time compared to manually designed STGNN models and NAS-based STGNN baselines, with only a minor accuracy drop (approximately 1.6%p). In addition, SearchLight reduces architecture searching time an average of $\times 29.5$ compared to existing NAS-based STGNNs, which suggests the potential of the proposed framework for rapid model development and deployment. SearchLight successfully balances accuracy and resource efficiency, and erases trial-and-error effort on lightweight STGNN model development for various domains. Future work includes integrating training time into the optimization objective, addressing diverse hardware constraints, and extending the framework to federated and distributed learning environments.

APPENDIX
MATRIX NOTATIONS

appendix]sec:appendix:matrix-notations We describe several matrix notations shown in the paper.

Definition 7 (Matrix Element Notation): A matrix element $X_{i,j} \in \mathbb{R}$ is in i-th row and j-th column of a matrix $X \in \mathbb{R}^{I \times J}$ where $0 \leq i < I$ and $0 \leq j < J$.

The matrix element notation of Definition 9 can be stretched to represent a submatrix by the colon notation of Definition 8. Here, we draw boxes to distinguish between other symbols and the colon notations.

Definition 8 (Colon Notation): A notation with a mix of colons and integers means a continuous range of integers. Given two integers n and m such that $0 \leq n < m \leq l$, $\boxed{n{:}m} := \{k | n \leq k < m\} = \{n, n+1, \ldots, m-1\}$. If any integers of the colon notation are omitted, the missing part is considered as the minimum or maximum possible integer, so that $\boxed{n{:}} := \{k | n \leq k < l\}$, $\boxed{{:}m} := \{k | 0 \leq k < m\}$ and $\boxed{:} := \{k | 0 \leq k < l\}$.

Finally, we use Definition 7 and Definition 8 to express a submatrix.

Definition 9 (Submatrix Notation): The submatrix $X_{a:b,c:d} \in \mathbb{R}^{(b-a)\times(d-c)}$ is the continuous range where row i is in $a \leq i < b$ and column j is in $c \leq j < d$ of matrix $X \in \mathbb{R}^{I \times J}$, such that $0 \leq a < b \leq I$ and $0 \leq c < d \leq J$.

REFERENCES

[1] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2017, *arXiv:1707.01926*.

[2] L. Chaolong, C. Zhen, Z. Wenming, X. Chunyan, and Y. Jian, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, Apr. 2018, pp. 7444–7452.

[3] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 3634–3640.

[4] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020.

[5] J. Tang, T. Qian, S. Liu, S. Du, J. Hu, and T. Li, "Spatio-temporal latent graph structure learning for traffic forecasting," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–8.

[6] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial–temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 922–929.

[7] T. Kwon, J. Ko, J. Jung, and K. Shin, "NeuKron: Constant-size lossy compression of sparse reorderable matrices and tensors," in *Proc. ACM Web Conf.*, Austin, TX, USA, Apr. 2023, pp. 71–81.

[8] T. Kwon, J. Ko, J. Jung, and K. Shin, "TensorCodec: Compact lossy compression of tensors without strong data assumptions," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Shanghai, China, Dec. 2023, pp. 229–238.

[9] J. Ko, T. Kwon, J. Jung, and K. Shin, "ELiCiT: Effective and lightweight lossy compression of tensors," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Abu Dhabi, United Arab Emirates, Dec. 2024, pp. 171–180.

[10] T. Wang, N. Xu, K. Chen, and W. Lin, "End-to-end video instance segmentation via spatial–temporal graph neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10777–10786.

[11] D. Cao, Y. Wang, J. Duan, C. Zhang, X. Zhu, C. Huang, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Spectral temporal graph neural network for multivariate time-series forecasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 17766–17778.

[12] M. Davidson and D. Moodley, "ST-GNNs for weather prediction in South Africa," in *Proc. Southern Afr. Conf. Artif. Intell. Res.*, Cham, Switzerland: Springer, 2022, pp. 93–107.

[13] S. Wang, X. Li, G. Liao, J. Liu, C. Liao, M. Liu, J. Liao, and L. Liu, "A spatio-temporal graph neural network for fall prediction with inertial sensors," *Knowl.-Based Syst.*, vol. 293, Jun. 2024, Art. no. 111709.

[14] X. Li, B. Qian, J. Wei, A. Li, X. Liu, and Q. Zheng, "Classify EEG and reveal latent graph structure with spatio-temporal graph convolutional neural network," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 389–398.

[15] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "GeoMAN: Multi-level attention networks for geo-sensory time series prediction," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 3428–3434.

[16] H. J. Douglas, *Time Series Analysis*. Princeton, NJ, USA: Princeton Univ. Press, 1994.

[17] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results," *J. Transp. Eng.*, vol. 129, no. 6, pp. 664–672, Nov. 2003.

[18] Q. Qi, L. Zhang, J. Wang, H. Sun, Z. Zhuang, J. Liao, and F. R. Yu, "Scalable parallel task scheduling for autonomous driving using multi-task deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 11, pp. 13861–13874, Nov. 2020.

[19] R. Mishra and H. Gupta, "Transforming large-size to lightweight deep neural networks for IoT applications," *ACM Comput. Surv.*, vol. 55, no. 11, pp. 1–35, Nov. 2023.

[20] G. Li, S. Zhong, X. Deng, L. Xiang, S.-H.-G. Chan, R. Li, Y. Liu, M. Zhang, C.-C. Hung, and W.-C. Peng, "A lightweight and accurate spatial–temporal transformer for traffic forecasting," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 10967–10980, Nov. 2023.

[21] P. Wang, H. Zhang, S. Cheng, T. Zhang, F. Lu, and S. Wu, "A lightweight spatiotemporal graph dilated convolutional network for urban sensor state prediction," *Sustain. Cities Soc.*, vol. 101, Feb. 2024, Art. no. 105105.

[22] J. Han, W. Zhang, H. Liu, T. Tao, N. Tan, and H. Xiong, "BigST: Linear complexity spatio-temporal graph neural network for traffic forecasting on large-scale road networks," *Proc. VLDB Endowment*, vol. 17, no. 5, pp. 1081–1090, Jan. 2024.

[23] K. K. Bhaumik, F. F. Niloy, S. Mahmud, and S. S. Woo, "STLGRU: Spatio-temporal lightweight graph GRU for traffic flow prediction," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*, 2024, pp. 288–299.

[24] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "AutoCTS: Automated correlated time series forecasting," *Proc. VLDB Endowment*, vol. 15, no. 4, pp. 971–983, Dec. 2021.

[25] S. Fang, C. Zhang, S. Xiang, and C. Pan, "AutoMSNet: Multi-source spatio-temporal network via automatic neural architecture search for traffic flow prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 3, pp. 2827–2841, Mar. 2023.

[26] T. Li, J. Zhang, K. Bao, Y. Liang, Y. Li, and Y. Zheng, "AutoST: Efficient neural architecture search for spatio-temporal prediction," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 794–802.

[27] Z. Pan, S. Ke, X. Yang, Y. Liang, Y. Yu, J. Zhang, and Y. Zheng, "AutoSTG: Neural architecture search for predictions of spatio-temporal graph," in *Proc. Web Conf.*, Apr. 2021, pp. 1846–1855.

[28] S. Ke, Z. Pan, T. He, Y. Liang, J. Zhang, and Y. Zheng, "AutoSTG+: An automatic framework to discover the optimal network for spatio-temporal graph prediction," *Artif. Intell.*, vol. 318, May 2023, Art. no. 103899.

[29] Z. Xu, Y. Li, and Q. Yang, "Understanding and simplifying architecture search in spatio-temporal graph neural networks," *Trans. Mach. Learn. Res.*, pp. 1–19, Jan. 2023.

[30] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.

[31] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 17804–17815.

[32] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for deep spatial–temporal graph modeling," 2019, *arXiv:1906.00121*.

[33] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," 2016, *arXiv:1609.03499*.

[34] S. Guo, Y. Lin, H. Wan, X. Li, and G. Cong, "Learning dynamics and heterogeneity of spatial–temporal graph data for traffic forecasting," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5415–5428, Nov. 2022.

[35] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," 2017, *arXiv:1709.04875*.

[36] G. Jin, Y. Liang, Y. Fang, Z. Shao, J. Huang, J. Zhang, and Y. Zheng, "Spatio-temporal graph neural networks for predictive learning in urban computing: A survey," 2023, *arXiv:2303.14483*.

[37] Z. Al Sahili and M. Awad, "Spatio-temporal graph neural networks: A survey," 2023, *arXiv:2301.10569*.

[38] J.-w. Lee and J. Jung, "Time-aware random walk diffusion to improve dynamic graph learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 37, 2023, pp. 8473–8481.

[39] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Dec. 2008.

[40] L. Hällman, "The rolling window method: Precisions of financial forecasting," Ph.D. dissertation, School Eng. Sci., Royal Inst. Technol., Stockholm, Sweden, 2017.

[41] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[42] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016.

[43] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[44] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[45] C. Liu, B. Zoph, M. Neumann, J. Shlens, H. Wei, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–35.

[46] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.

[47] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[48] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[50] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[51] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016.

[52] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," 2018, *arXiv:1810.05997*.

[53] K. Shin, J. Jung, S. Lee, and U. Kang, "BEAR: Block elimination approach for random walk with restart on large graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2015, pp. 1571–1585.

[54] J. Jung, N. Park, S. Lee, and U. Kang, "BePI: Fast and memory-efficient method for billion-scale random walk with restart," in *Proc. ACM Int. Conf. Manage. Data*, May 2017, pp. 789–804.

[55] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "MONAS: Multi-objective neural architecture search using reinforcement learning," 2018, *arXiv:1806.10332*.

[56] P. Achararit, M. A. Hanif, R. V. W. Putra, M. Shafique, and Y. Hara-Azumi, "APNAS: Accuracy-and-Performance-Aware neural architecture search for neural hardware accelerators," *IEEE Access*, vol. 8, pp. 165319–165334, 2020.

[57] C. Wang, H. Wang, G. Feng, and F. Geng, "Multi-objective neural architecture search based on diverse structures and adaptive recommendation," 2020, *arXiv:2007.02749*.

[58] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.

[59] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[60] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 419–427.

[61] S. Kim and H. Kim, "A new metric of absolute percentage error for intermittent demand forecasts," *Int. J. Forecasting*, vol. 32, no. 3, pp. 669–679, Jul. 2016.

[62] Y. Zheng, X. Yi, M. Li, R. Li, Z. Shan, E. Chang, and T. Li, "Forecasting fine-grained air quality based on big data," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 2267–2276.

[63] Korea National Oil Corporation. *Opinet*. Website. Accessed: 2024. [Online]. Available: https://www.opinet.co.kr/

[64] A. Smith, N. Lott, and R. Vose, "The integrated surface database: Recent developments and partnerships," *Bull. Amer. Meteorolog. Soc.*, vol. 92, no. 6, pp. 704–708, Jun. 2011.

[65] K. Uzel, S. Kato, and M. Zimmer, "A set of hub neurons and non-local connectivity features support global brain dynamics in C. Elegans," *Current Biol.*, vol. 32, no. 16, pp. 3443–3459.e8, Aug. 2022.

[66] *The Freeway Performance Measurement (PEMS)*, Website, California Department of Transportation, Sacramento, CA, USA, 2018.

HEEYONG YOON received the B.Sc. degree from Daegu Gyeongbuk Institute of Science and Technology (DGIST), in 2018, where he is currently pursuing the Ph.D. degree with the Department of Information and Communication Engineering. His research interests include distributed computing, high-performance GPU systems, time series data, and graph processing.

JINHONG JUNG received the B.S. degree in computer science and engineering from Jeonbuk National University, the M.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), and the Ph.D. degree from the Department of Computer Science and Engineering, Seoul National University, South Korea, in 2020, where he was advised by Prof. U. Kang and supported by the Global Ph.D. Fellowship. He is currently an Assistant Professor with the School of Software, Soongsil University, South Korea. Previously, he was an Assistant Professor with Jeonbuk National University and a Postdoctoral Researcher with Seoul National University. His research interests include graph machine learning, large-scale data analytics, and recommender systems. He won the Best Student Paper Runner-Up Award at ICDM 2023.

KANG-WOOK CHON received the Ph.D. degree in computer science from Daegu Gyeongbuk Institute of Science and Technology (DGIST), in 2018. He was a Software Engineer developing big data systems with SKT, from 2018 to 2020. He was a Senior Researcher with the National Supercomputing Center, KISTI, from 2020 to 2023. He is currently an Assistant Professor with the School of Computer Engineering, Korea University of Technology and Education (KOREATECH). His research interests include scalable data mining, machine learning, and bioinformatics using parallel computing and distributed computing.

MIN-SOO KIM (Senior Member, IEEE) received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), in 2006. He was Postdoctoral Fellow in data mining with UIUC. He was with IBM Almaden Research for a project to develop the IBM Smart Analytics Optimizer for DB2 for z/OS. From 2011 to 2020, he was with the Department of Information and Communication Engineering, Daegu Gyeongbuk Institute of Science and Technology (DGIST). He is currently with the School of Computing, KAIST. His research interests include databases, data mining, machine learning, and bioinformatics.