Dynamic Heterogeneous Graph Attention Neural Architecture Search (Appendix)

Zeyang Zhang[1*], **Ziwei Zhang**[1], **Xin Wang**[1†], **Yijian Qin**[1], **Zhou Qin**[2], **Wenwu Zhu**[1†]
[1]Tsinghua University, [2]Alibaba Group
{zy-zhang20,qinyj19}@mails.tsinghua.edu.cn, qinzhou.qinzhou@alibaba-inc.com,
{zwzhang,xin_wang,wwzhu}@tsinghua.edu.cn

A Notations

We summarize the notations and descriptions in Table 1.

B Special Cases of the Search Space

As discussed in Section 3.2, our proposed search space covers several classic GNN architectures. In this section, we illustrate how these architectures can be recovered in our search space. We omit the details regarding attention implementation (e.g., multiplicative or additive) since they can be extended easily.

Temporal self-attention (Fan et al. 2022): For any node type c_n, we can add a relation type c_{self} to indicate edges of self-loops. Temporal self-attention can be recovered when the localization space is

$$\mathbf{A}^{Lo}_{t,t',r} = \begin{cases} 1, & r = c_{self} \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

Masked temporal self-attention (Xue et al. 2020; Sankar et al. 2020): Masked temporal self-attention forbids reverse-time connections in temporal self-attention, i.e.,

$$\mathbf{A}^{Lo}_{t,t',r} = \begin{cases} 1, & r = c_{self} \text{ and } t >= t' \\ 0, & \text{otherwise} \end{cases}. \tag{2}$$

GAT (Veličković et al. 2018): Our search space can cover the representative static attention-based GNN by only containing spatial connections without any temporal connections:

$$\begin{gathered} \mathbf{A}^{Lo}_{t,t',r} = \begin{cases} 1, & t = t' \\ 0, & \text{otherwise} \end{cases}, \\ \mathbf{A}^{N}_{t,n} = 0, \mathbf{A}^{R}_{t,r} = 0 \quad \forall t, n, r. \end{gathered} \tag{3}$$

HGT (Hu et al. 2020b): Our search space can cover this representative heterogeneous GNN as:

$$\begin{gathered} \mathbf{A}^{Lo}_{t,t',r} = \begin{cases} 1, & t = t' \\ 0, & \text{otherwise} \end{cases}, \\ \mathbf{A}^{N}_{t,n} = n, \mathbf{A}^{R}_{t,r} = n \quad \forall t, n, r. \end{gathered} \tag{4}$$

Multi-layer perceptron: Our space can also cover architectures that do not consider graph structure and time slices as:

$$\mathbf{A}^{Lo}_{t,t',r} = \begin{cases} 1, & r = c_{self} \text{ and } t = t' \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Notice that in this architecture, the model can assign different MLP weights for different node types.

Identity Mapping. When the localization result $\mathbf{A}^{Lo}$ is full of zeros, the representations in the previous layer are directly fed into the next layer, i.e., identity mapping, which is necessary to build skip-connections in GNNs.

With slight changes to the attention mechanism, **DHGAS** can also include **DyHATR** (Xue et al. 2020) and **HTGNN** (Fan et al. 2022). Specifically, we need to alter the unified graph attention in DHGA into hierarchical attention, i.e., first aggregate messages w.r.t each relation type and then aggregate messages from each relation type to calculate the target node representation. Then, we can instantiate static heterogeneous layers and temporal self-attention layers to implement similar to other methods mentioned above.

C Additional Experiments

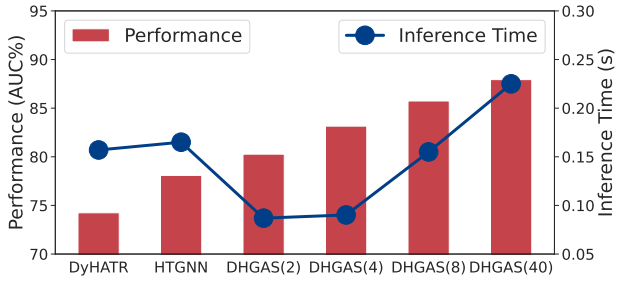
Tradeoff between the computational budgets and performance. To verify that **DHGAS** can search architectures tailored to the datasets as well as balance the computational budgets and performance, we compare the searched architectures with different budgets K_{Lo} on Aminer and Ecomm datasets in terms of inference time and performance. From Figure 1, we can see that as we gradually add computational budgets K_{Lo}, **DHGAS** can search architectures with better performance. Empirically, we find that with lower computational budgets, **DHGAS** can also find competitive architectures. For example, as shown in Figure 4 and Figure 5, under budgets of $K_{Lo} = 16$, **DHGAS** finds an architecture with AUC 84.2% (which still outperforms the best baseline with AUC 82.3%), while being 44% faster in the inference time than the architecture searched under budgets of $K_{Lo} = 64$ (with AUC 86.5%).

Supernet training and model inference time. We list the inference time (in seconds) of the search models by our method **DHGAS** and static heterogeneous NAS baseline DiffMG, as well as representative hand-designed dynamic heterogeneous GNN baseline HTGNN and repre-

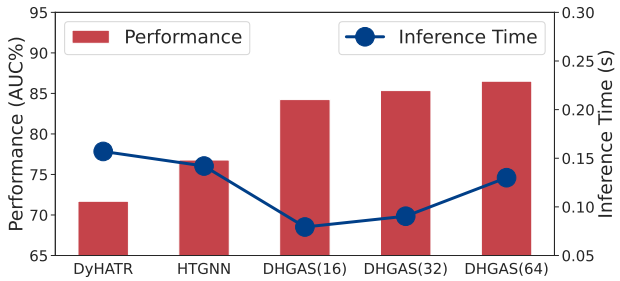
[*]This work was done during author's internship at Alibaba Group
[†]Corresponding authors

Table 1: A summary of notations

Notations	Descriptions
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A graph with the node set and the edge set
ϕ_n, ϕ_e	The node and edge type mapping function
$\mathcal{C}_n, \mathcal{C}_e$	The node type set and the edge type set
t, T	A time stamp and the number of time stamps
$\mathcal{E}_r^t$	Edges with relation type r at time stamp t
$\mathcal{N}_r^t(u)$	The neighborhood of node u with relation type r at time stamp t
$\mathbf{h}_u^t$	The representation of node u at time stamp t
d	The dimensionality of node representation
$\mathbf{q}_u^t, \mathcal{F}_{q,c,t}^N$	The query vector and node mapping function of node u with node type c at time stamp t
$\mathbf{k}_u^t, \mathcal{F}_{k,c,t}^N$	The key vector and node mapping function of node u with node type c at time stamp t
$\mathbf{v}_u^t, \mathcal{F}_{v,c,t}^N$	The value vector and node mapping function of node u with node type c at time stamp t
$\mathcal{F}_{c,\Delta t}^R$	The relation mapping function for relation type c at time interval Δt
$\alpha_{u,v}$	The attention weight between node u and v
$\mathcal{A}$	The overall architecture space
$\mathcal{A}^{Lo}, \mathcal{A}^{Pa}$	The localization and the parameterization search space
K_{Lo}, K_N, K_R	The search space constraint hyper-parameters



(a) Aminer



(b) Ecomm

Figure 1: Comparison of searched architectures under different computational budgets K_{Lo} in terms of inference time and performance, where **DHGAS**(k) refers to the architecture searched by **DHGAS** with budget $K_{Lo} = k$.

Table 2: The inference time (in seconds) of the search models of different methods

Type	Method	Aminer	Ecomm	Yelp	Drugs	COVID-19
Static Heterogenous GNN	RGCN	0.005	0.008	0.013	0.017	0.006
Static Heterogenous GNAS	DiffMG	0.006	0.007	0.012	0.015	0.007
Dynamic Heterogenous GNN	HTGNN	0.074	0.063	0.216	0.193	0.084
Dynamic Heterogenous GNAS	DHGAS	0.086	0.057	0.211	0.184	0.106

Table 3: The supernet training time (in seconds) of different methods

Type	Method	Aminer	Ecomm	Yelp	Drugs	COVID-19
Static Heterogeneous GNAS	DiffMG	19	38	20	34	1540
Dynamic Heterogeneous GNAS	DHGAS	351	102	148	186	11,089

sentative hand-designed static heterogeneous GNN baseline RGCN. The time is tested using a single GPU. As shown in Table 2, static GNNs are faster than dynamic GNNs, mainly because they neglect the time dimension, inevitably leading to their unsatisfactory performance on dynamic graphs for failing to capture the temporal patterns. Compared to hand-designed dynamic heterogeneous GNN HTGNN, **DHGAS** shows comparable inference time, indicating that our method does not put extra complexity to tackle dynamic information, while capable of more flexible designing models and better prediction. Next, we also report the supernet training time (in seconds) of our dynamic heterogeneous GNAS **DHGAS** and static heterogeneous GNAS baseline DiffMG. As DiffMG is specially designed for static heterogeneous graphs, its training time is lower since it does not tackle dynamic information and ignores temporal information during training. On the other hand, **DHGAS** provides automatic DHGNN designing and better performance by jointly considering dynamics and heterogeneity with a reasonable increase in computation.

Table 4: Comparison with β-DARTS on Aminer dataset with **DHGAS** search space.(AUC%)

K_{Lo}	**4**	**8**	**10**	**20**	**40**
β-DARTS	73.87	80.16	82.81	81.13	86.30
DHGAS	82.16	84.55	83.92	83.69	87.98

Comparisons with β-DARTS. To improve the robust-

ness and generalization ability of DARTS, β-DARTS (Ye et al. 2022) proposes a regularization to keep the value and variance of activated architecture parameters from too large. As shown in the Table 4, β-DARTS (Ye et al. 2022) acts as a strong baseline and shows improvements over DARTS in most cases. Nevertheless, our method still shows better performance, which we attribute to its specific design for our proposed localization and parameterization search space. Besides, since β-DARTS is general and orthogonal to **DHGAS**, **DHGAS** may be further enhanced when combined with these powerful techniques, which we will leave as future explorations.

Results on the million-scale OGBN-MAG dataset. To further verify the effectiveness of **DHGAS**, we add additional experiments on a significantly larger dataset OGBN-MAG (Hu et al. 2020a). The original OGBN-MAG dataset is a static heterogeneous network composed of a subset of the Microsoft Academic Graph (Wang et al. 2020). We follow (Fan et al. 2022) to extract a dynamic heterogeneous graph consisting of 10 graph slices spanning from 2010 to 2019, and the task is to predict whether a pair of authors will co-author in the next year. The statistics of OGBN-MAG are summarized in Table 5.

Table 5: Statistics of OGBN-MAG dataset.

# Nodes for each type	#Author: 17,764 #Paper: 282,039 #Field: 34,601 #Institution: 2,276
# Edges for each type	#Author-Writes-Paper: 2,061,677 #Paper-Cites-Paper: 2,377,654 #Paper-HasTopic-Field: 289,376 #Author-AffiliatedWith-Institution: 40,307
# Time Slices	10
Node Feature Dimension	128

Following (Fan et al. 2022), we adopt Adam optimizer with a learning rate set to 5e-3, weight decay set to 5e-4, and set hidden embedding dimension to 32, patience of early stopping strategy to 50, maximum training epochs to 500, GNN layer to 2, number of heads to 4, time window length to 3. Other settings are the same as described in link prediction tasks. We compare representative static homogeneous GNN GAT (Veličković et al. 2018), static heterogeneous GNN HGT (Hu et al. 2020b), and dynamic heterogeneous GNN HTGNN (Fan et al. 2022).

Table 6: Comparisons with representative methods on the million-scale OGBN-MAG dataset.

Type	Method	AUC%
Static Homogeneous	GAT	80.23±2.07
Static Heterogeneous	HGT	85.30±1.20
Dynamic Heterogeneous	HTGNN	91.01±0.77
Dynamic Heterogeneous	DHGAS(ours)	93.83±0.08

As shown in Table 6, **DHGAS** outperforms all the baseline methods, which we attribute to the following two points. First, **DHGAS**'s is designed to tackle graph heterogeneity and dynamics simultaneously, which can fully mine the information inside dynamic heterogeneous graph data, while the static methods GAT and HGT have information loss due to their ignorance of graph dynamics. Second, as the **DHGAS** search space is general and flexible to include GAT, HGT, and HTGNN as detailed in Section B, **DHGAS** can automate and tailor the design of DHGNN for different datasets.

D Details for Reproducibility

D.1 DHGAS Implementation Details

In this section, we describe the details of the implementation of our proposed **DHGAS**.

Since node features of different types may have various dimensionalities and semantics, following HGT (Hu et al. 2020b), we first conduct node-type-aware fully-connected layers to map all node features into the hidden space, which can be formulated as

$$\text{FC}(\mathbf{h}_u) = \text{Activation}(\mathbf{W}_{\phi_n(u)}\mathbf{h}_u + \mathbf{b}_{\phi(u)}). \tag{6}$$

Following the preprocessing layer, our model stacks multiple searched layers and uses the output of the last layer as the final representation for downstream tasks.

For Eq. (7), we implement the update function using node-type-aware fully-connected layers and layer normalization as:

$$\text{Update}(\mathbf{h}_u, \mathbf{m}_u) = \text{LayerNorm}_{\phi(u)}\left(\mathbf{h}_u + \text{GeLU}(\text{FC}(\mathbf{m}_u))\right). \tag{7}$$

On the COVID-19 dataset, we remove layer normalization as HTGNN (Xue et al. 2020) since the scales of features are important for this task.

D.2 Datasets

We use the following 5 dynamic heterogeneous graph datasets, and the statistics are summarized in Table 7.

- **Aminer**[1] (Ji et al. 2021) is an academic citation dataset for papers that were published during 1990-2006. The dataset has three types of nodes (paper, author and venue), and two types of relations (paper-publish-venue and author-writer-paper). We separate time slices using the publication year.
- **Ecomm**[2] (Xue et al. 2020) is a recommendation dataset, recording shopping behaviors of users from 10th June 2019 to 20th June 2019. It has two types of nodes (user and item) and four types of relations (user-click-item, user-buy-item, user-(add-to)-cart-item and user-(add-to)-favorite-item).
- **Yelp**[3] (Ji et al. 2021) is a business review dataset, containing user reviews and tips on business. Following (Ji et al. 2021), we consider interactions of three categories of business including "American (New) Food", "Fast Food" and "Sushi" from January 2012 to December 2012.

[1] https://www.aminer.cn/collaboration.

[2] https://tianchi.aliyun.com/competition/entrance/231719

[3] https://www.yelp.com/dataset

Table 7: Statistics of datasets.

Dataset	# Total Time Slices	# Nodes for each type	# Edges for each type
Aminer	16	# Paper : 18,464 # Author : 23,035 # Venue : 22	# Paper-publish-Venue : 18,464 # Author-write-Paper : 52,545
Ecomm	10	# User : 1,476 # Item : 34,505	# User-click-Item : 57,917 # User-buy-Item : 5,529 # User-cart-Item : 15,066 # User-favorite-Item: 6,701
Yelp	12	# User : 55,702 # Business : 12,524	# User-review-Business : 87,846 # User-tip-Business : 35,508
Drugs	15	# Seller : 6,970 # Buyer : 29,481 # Product : 13,252	# Seller-sell-Product: 13,252 # Buyer-inquire-Product: 34,086 # Buyer-vote-Product: 5,632 # Buyer-buy-Product: 10,635 # Seller-similar-Seller : 415 # Buyer-follow-Seller: 1,197 # Buyer-favorite-Product: 7,621
COVID-19	304	# State : 54 # County : 3,223	# State-near-State : 269 # State-includes-County : 3,141 # County-near-County : 22,176

The dataset contains two types of nodes (users and business) and two types of edges (user-tip-business and user-review-business).

- **Drugs**[4] is an e-commerce risk management dataset, which records shopping behaviors of users from 1st December 2021 to 30th December 2021. We collect data every two days, which results in $T = 15$ time slices. The dataset contains three types of nodes (seller, buyer and product) and seven types of relations (seller-sell-product, buyer-buy-product, buyer-vote-product, buyer (-add-to)-favorite-product, buyer-follow-seller, buyer-inquire(-about)-product, seller-similar(-to)-seller).
- **COVID-19**[5] (Fan et al. 2022) is an epidemic disease dataset, containing daily new COVID-19 cases in the US. Following HTGNN (Fan et al. 2022), we extract graph slices from 05/01/2020 to 02/28/2021. The dataset contains two types of nodes (state and county) and three types of relations (state-includes-county, state-near-state and county-near-county).

D.3 Baselines

We compare with state-of-the-art hand-designed DHGNNs and graph NAS models.

Hand-designed Models. We compare the following seven models covering static/dynamic and homogeneous/heterogeneous GNNs:

- **GCN** (Kipf and Welling 2017): a representative static homogeneous GNN aggregating neighbors using degree normalized weights.
- **GAT** (Veličković et al. 2018): a representative static homogeneous GNN aggregating neighbors using the attention mechanism.
- **RGCN** (Schlichtkrull et al. 2018): a static heterogeneous GNN that assigns different parameterizations for different relation types.
- **HGT** (Hu et al. 2020b): a static heterogeneous GNN adopting mutual attention and different attention parameterization for different node and relation types.
- **DyHATR**(Xue et al. 2020): a dynamic heterogeneous GNN that uses hierarchical attention and temporal self-attention to capture heterogeneous and temporal information.
- **HGT+** (Hu et al. 2020b): a dynamic heterogeneous GNN that extends HGT by utilizing the relative temporal encoding to model temporal information.
- **HTGNN** (Fan et al. 2022): a dynamic heterogeneous GNN that uses hierarchical attention and temporal self-attention iteratively to capture complex dynamic heterogeneous information.

Graph NAS Models. We compare to two state-of-the-art graph NAS methods:

- **GraphNAS** (Gao et al. 2020): a representative static homogeneous graph NAS method. GraphNAS designs a search space including representative GNNs like GCN, GAT, *etc*, and adopts reinforcement learning to explore the search space.
- **DiffMG** (Ding et al. 2021): a representative static heterogeneous graph NAS method. DiffMG automates the static heterogenous GNN designs by searching meta-paths used

[4]Collected from Alibaba.com

[5]https://coronavirus.1point3acres.com/en

by GCN and exploring the search space with its specially designed differentiable search algorithm.

D.4 Training Protocol and Hyper-parameters

Training Protocol. For static graph models, we follow the literature and transform dynamic graphs into static ones by merging all graph slices. For dynamic models, we follow (Sankar et al. 2020) and sample T consecutive graph slices as the training data. For homogeneous models, we transform heterogeneous graphs into homogeneous ones by ignoring the node and edge types. For NAS methods including our proposed **DHGAS**, we first obtain the output architecture and then retrain and evaluate the architecture the same way as hand-designed models.

Hyper-parameters For all baselines and datasets, we choose the number of message-passing layers in $\{1, 2, 3\}$ and the number of attention heads in $\{1, 2, 4\}$ with the hidden representation dimensionality $d = 8$. For dynamic models, we choose the number of time slices T from $\{\lfloor T_{max}/3 \rfloor, \lfloor 2T_{max}/3 \rfloor, T_{max}\}$ for datasets except for COVID-19, where T_{max} is the total number of graph slices in the dataset that can be used for training. For COVID-19, we set the number of time slices as $T = 7$ following (Fan et al. 2022). Other hyper-parameters for baselines are kept the same as in the original paper. We adopt the Adam optimizer with a learning rate of 0.01 and set the early stopping round on the validation set as 10. We repeat all experiments 3 times and report the average results and standard deviations.

D.5 Task Setup and Loss Functions

In this section, we formulate the loss functions used in each task.

Link Prediction. For Aminer, the task is to predict links between author nodes, i.e., whether a pair of authors will coauthor a paper. We extract 32-dimensional paper features using word2vec (Mikolov et al. 2014). For other types of nodes, we assign one-hot encoding of node IDs as initial features. For Ecomm, the task is to predict whether a user will interact with an item, including click, buy, cart and favorite. The node features are also set as a one-hot encoding of node IDs. For both datasets, we use the last graph time slice for testing, the penultimate graph time slice for validation, and the rest for training. We randomly sample negative samples from nodes that do not have links, and the negative samples for validation and testing set are kept the same for all comparing methods. We use the Area under the ROC Curve (AUC) as the evaluation metric. For all baselines and our proposed method, we use the inner product on the two learned node representations to predict links and use cross-entropy as the loss function. We use the cross-entropy loss as follows:

$$\mathcal{L} = -\sum_{(u,v)\in\mathcal{E}^+} \log\sigma\left(\mathbf{h}_u^\top \mathbf{h}_v\right) - \sum_{(u',v')\in\mathcal{E}^-} \log\sigma\left(-\mathbf{h}_{u'}^\top \mathbf{h}_{v'}\right), \tag{8}$$

where $\mathcal{E}^+$ and $\mathcal{E}^-$ denote the set of positive edges and negative edges respectively, σ denotes the sigmoid function, and $\mathbf{h}_u$ and $\mathbf{h}_v$ are output node representations by the model.

Node Classification. For Yelp, the task is to classify the type of business nodes, i.e., a three-class classification problem. For Drugs, the task is to classify whether a seller sells drugs or not, i.e., a binary classification. We randomly split the target nodes into training, validation and testing with a ratio of 80%:10%:10%. We adopt metapath2vec (Dong, Chawla, and Swami 2017) to obtain 32-dimensional features for all types of nodes. We use the F1-score for Drugs and Macro-F1 for Yelp as the evaluation metric. For all the methods, we adopt a fully-connected layer with the softmax activation function as the classification layer and use cross-entropy as the loss function. We use the cross-entropy loss as follows:

$$\mathcal{L} = -\sum_{v\in\mathcal{V}_L}\sum_{c=1}^{C_Y} \mathbf{y}_v[c]\log\overline{\mathbf{y}}_v[c], \tag{9}$$

where $\mathcal{V}_L$ is the set of nodes with labels, $\mathbf{y}_v$ is a one-hot vector indicating the ground-truth label of node v and $\overline{\mathbf{y}}_v$ is the predicted label.

Node Regression. The task of node regression on COVID-19 is to predict the new daily cases. We split the dataset into a training, validation and testing set with a ratio of 80%:10%:10%. We use Mean Absolute Errors (MAE) as the evaluation metric. For all the methods, we adopt a fully connected layer with the ReLU activation function as the predictor and adopt MAE Loss as the loss function. Notice that in this dataset, only node features change over time. Therefore, for static baselines, we only use the information from the last time slice, i.e., use t to predict $t+1$. Besides, we omit the result for HGT+ since the time encoding technique cannot be applied here. We use the MAE loss as follows:

$$\mathcal{L} = \frac{1}{|\mathcal{V}_L|}\sum_{v\in\mathcal{V}_L} |y_v - \hat{y}_v|, \tag{10}$$

where $\mathcal{V}_L$ is the set of target nodes, y_v is the ground-truth label (an integer) of node v and $\hat{y}_v$ is the node v's regression value output by the model.

D.6 Configurations

Experiments on Aminer, Ecomm, Yelp and COVID-19 datasets are conducted with:

- Operating System: Ubuntu 18.04.1 LTS
- CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
- GPU: TITAN Xp with 12 GB of memory
- Software: Python 3.6.13, Cuda 10.2, PyTorch 1.8.2 (Paszke et al. 2019), PyTorch Geometric 2.0.3 (Fey and Lenssen 2019).

Experiments on Drugs dataset are conducted with:

- Operating System: Ubuntu18.04.6 LTS (Bionic Beaver)
- CPU: Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz
- GPU: NVIDIA Tesla V100 with 16 GB of memory
- Software: Python 3.6.12, Cuda 10.1, PyTorch 1.8.2 (Paszke et al. 2019), PyTorch Geometric 2.0.3 (Fey and Lenssen 2019).

D.7 Licenses

The licenses of the baselines and datasets are as follows:

- GNU Affero General Public License 3.0: COVID-19 [6]
- MIT License: DyHATR[7],HGT [8], PyTorch-Geometric [9]
- Apache License 2.0: GraphNAS[10], DARTS[11], Yelp[12].
- Unspecified license: Aminer[13], Ecomm[14], HTGNN[15], DiffMG[16]
- Other license: PyTorch[17]

E Visualization

In this section, we visualize several architectures searched with different computational budgets K_{Lo} on Aminer and Ecomm datasets. In each figure, the letter and number in the node denote node type and time respectively. The colors of the nodes and edges denote their choices of node and relation mapping functions. For example, in Figure 2, one yellow edge from yellow node 'a7' to green node 'c0' means that node type a at time 7 attends to node type c at time 0, and the attention is calculated by yellow node mapping function for a7, green node mapping function for c0 and yellow relation mapping function for their relation.

References

Ding, Y.; Yao, Q.; Zhao, H.; and Zhang, T. 2021. Diffmg: Differentiable meta graph search for heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 279–288.

Dong, Y.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 135–144.

Fan, Y.; Ju, M.; Zhang, C.; and Ye, Y. 2022. Heterogeneous Temporal Graph Neural Network. In *Proceedings of the 2022 SIAM International Conference on Data Mining*, 657–665.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; and Hu, Y. 2020. Graph Neural Architecture Search. In *IJCAI*, volume 20, 1403–1409.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020a. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 22118–22133. Curran Associates, Inc.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020b. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, 2704–2710.

Ji, Y.; Jia, T.; Fang, Y.; and Shi, C. 2021. Dynamic heterogeneous graph embedding via heterogeneous hawkes process. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 388–403.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2014. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Sankar, A.; Wu, Y.; Gou, L.; Zhang, W.; and Yang, H. 2020. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 519–527.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 593–607.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Wang, K.; Shen, Z.; Huang, C.; Wu, C.; Dong, Y.; and Kanakia, A. 2020. Microsoft Academic Graph: When experts are not enough. *Quant. Sci. Stud.*, 1(1): 396–413.

Xue, H.; Yang, L.; Jiang, W.; Wei, Y.; Hu, Y.; and Lin, Y. 2020. Modeling dynamic heterogeneous network for link prediction using hierarchical attention with temporal rnn. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 282–298.

Ye, P.; Li, B.; Li, Y.; Chen, T.; Fan, J.; and Ouyang, W. 2022. beta-DARTS: Beta-Decay Regularization for Differentiable Architecture Search. *arXiv preprint arXiv:2203.01665*.

[6] https://coronavirus.1point3acres.com/en/data

[7] https://github.com/skx300/DyHATR/blob/master/LICENSE

[8] https://github.com/acbull/pyHGT/blob/master/LICENSE

[9] https://github.com/pyg-team/pytorch_geometric/blob/master/LICENSE

[10] https://github.com/GraphNAS/GraphNAS/blob/master/LICENSE

[11] https://github.com/quark0/darts/blob/master/LICENSE

[12] https://www.yelp.com/dataset/

[13] https://www.aminer.cn/collaboration

[14] https://tianchi.aliyun.com/competition/entrance/231719

[15] https://github.com/YesLab-Code/HTGNN

[16] https://github.com/AutoML-Research/DiffMG

[17] https://github.com/pytorch/pytorch/blob/master/LICENSE

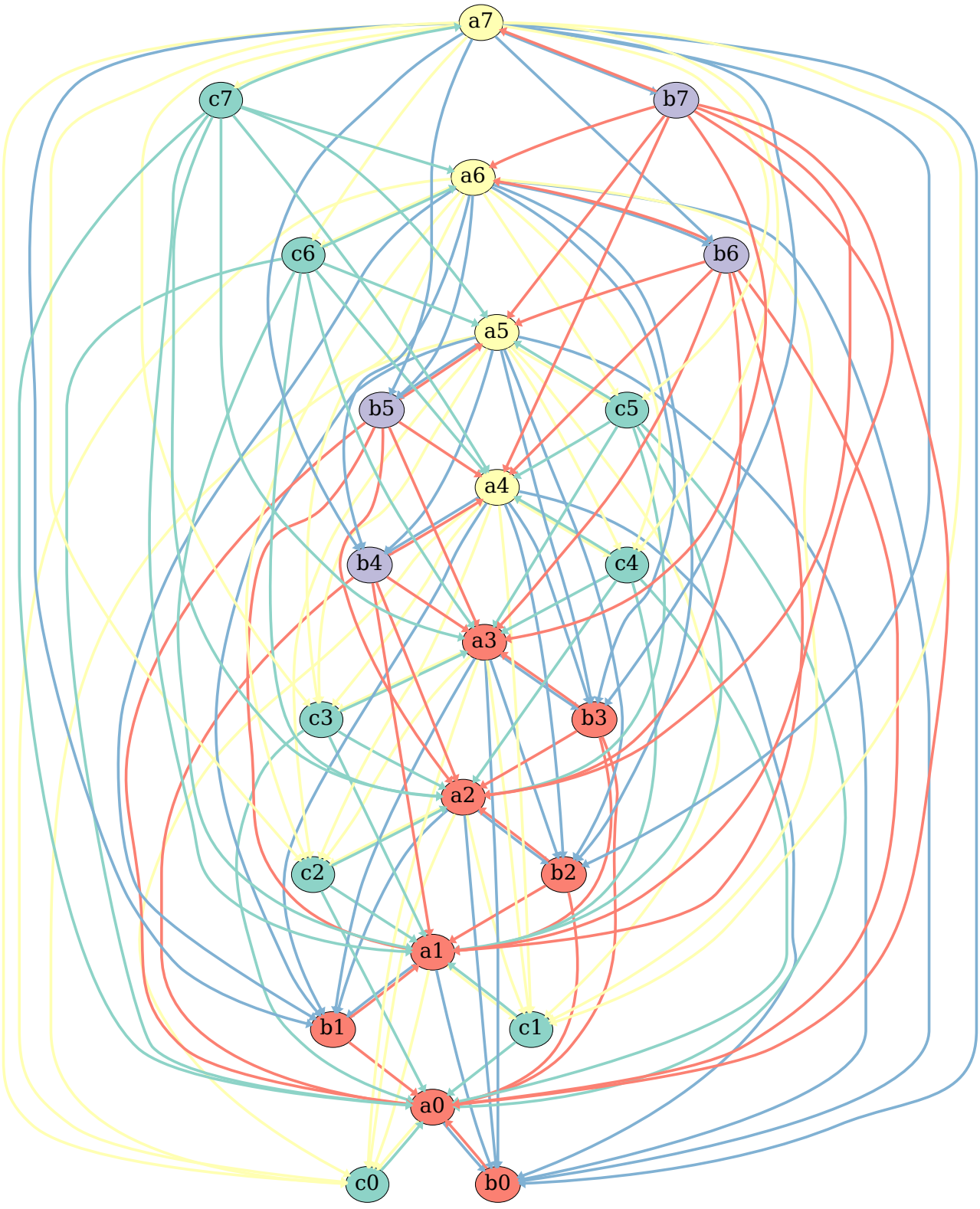


Figure 2: The searched architecture with $K_{Lo} = 40$ on Aminer dataset, which achieves performance of 87.9% AUC.

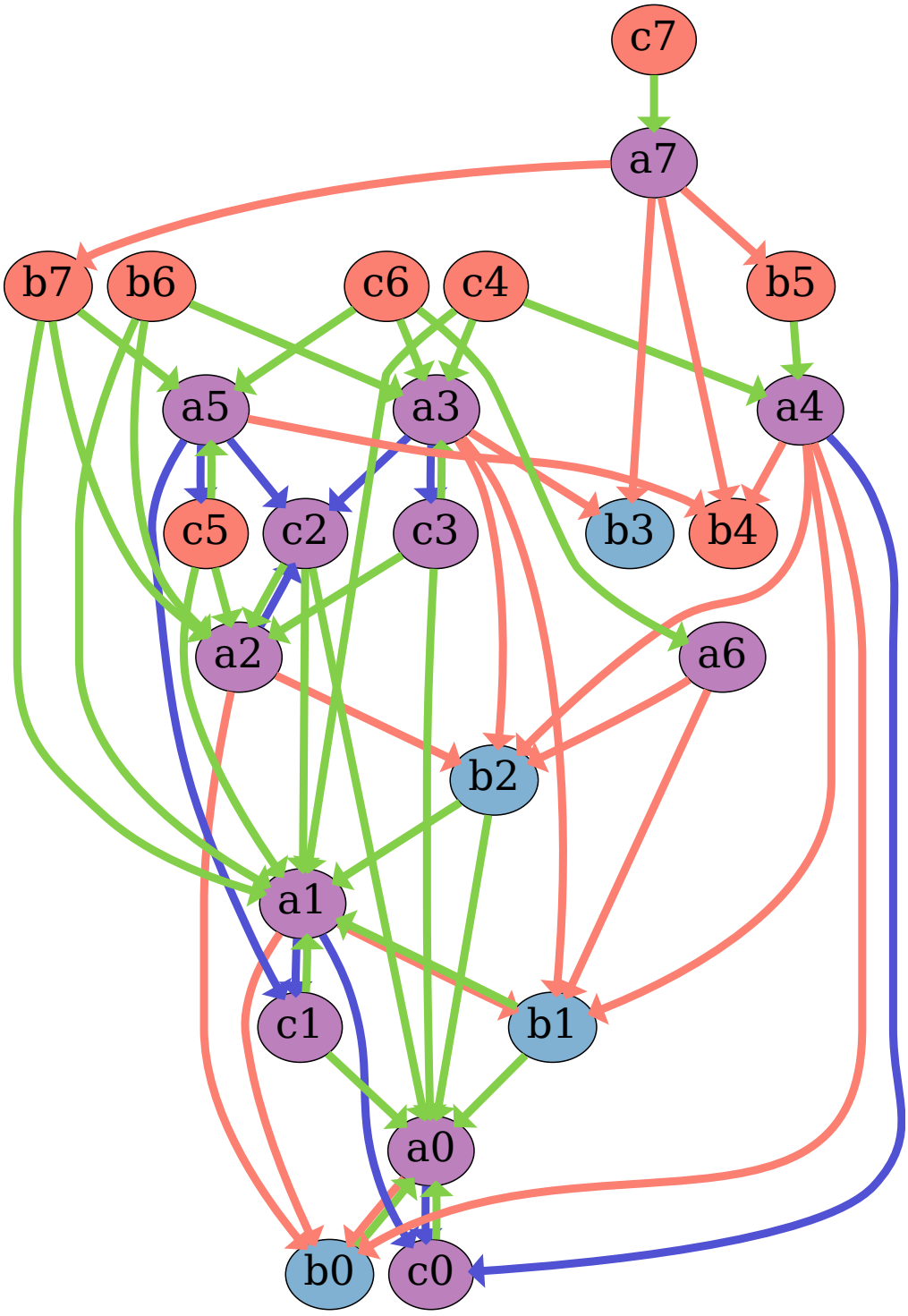


Figure 3: The searched architecture with $K_{Lo} = 8$ on Aminer dataset, which achieves performance of 85.4% AUC, while is nearly 31% faster in inference time than the searched architecture with $K_{Lo} = 40$.

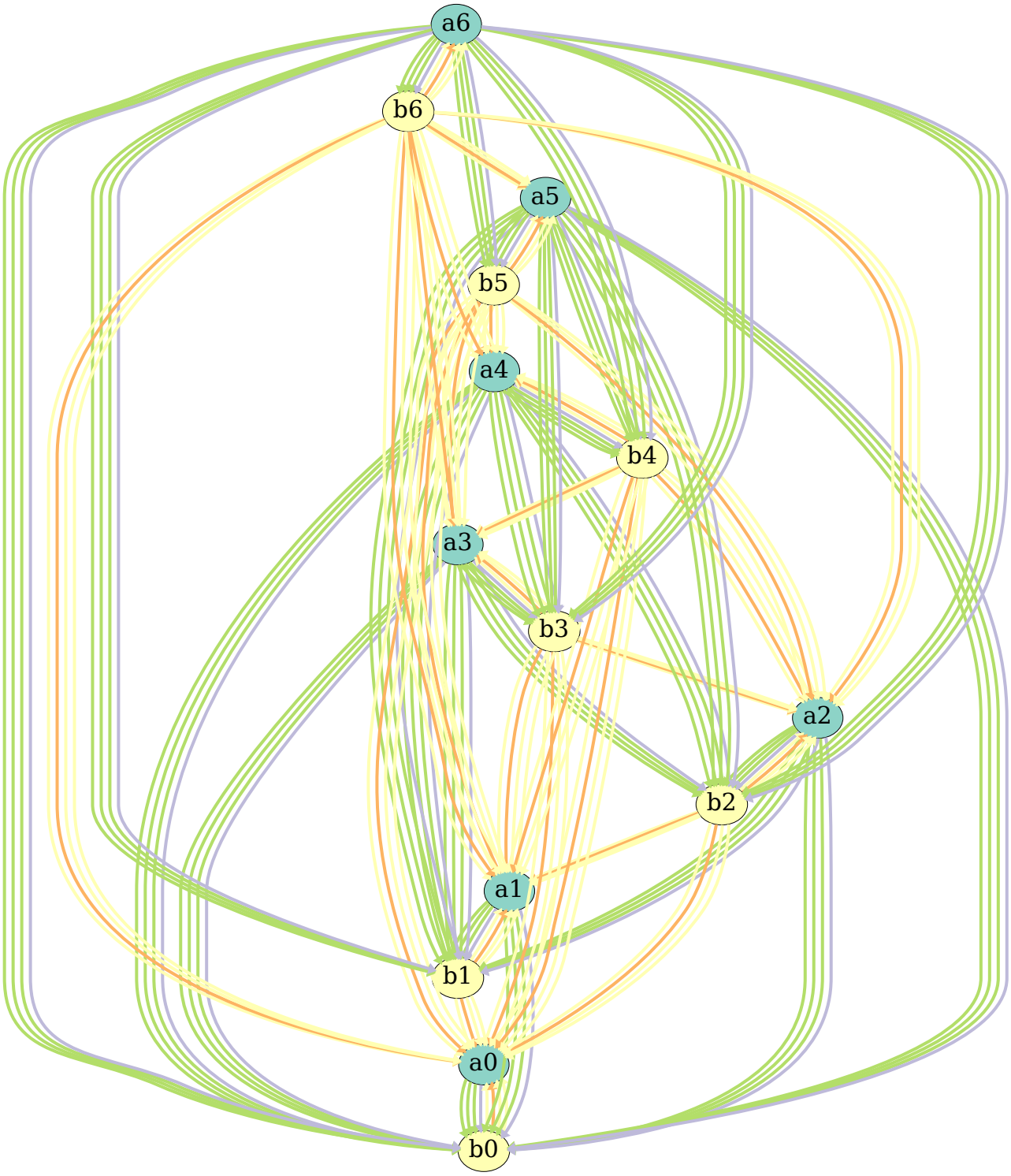


Figure 4: The searched architecture with $K_{Lo} = 64$ on Ecomm dataset, which achieves performance of 86.5% AUC.

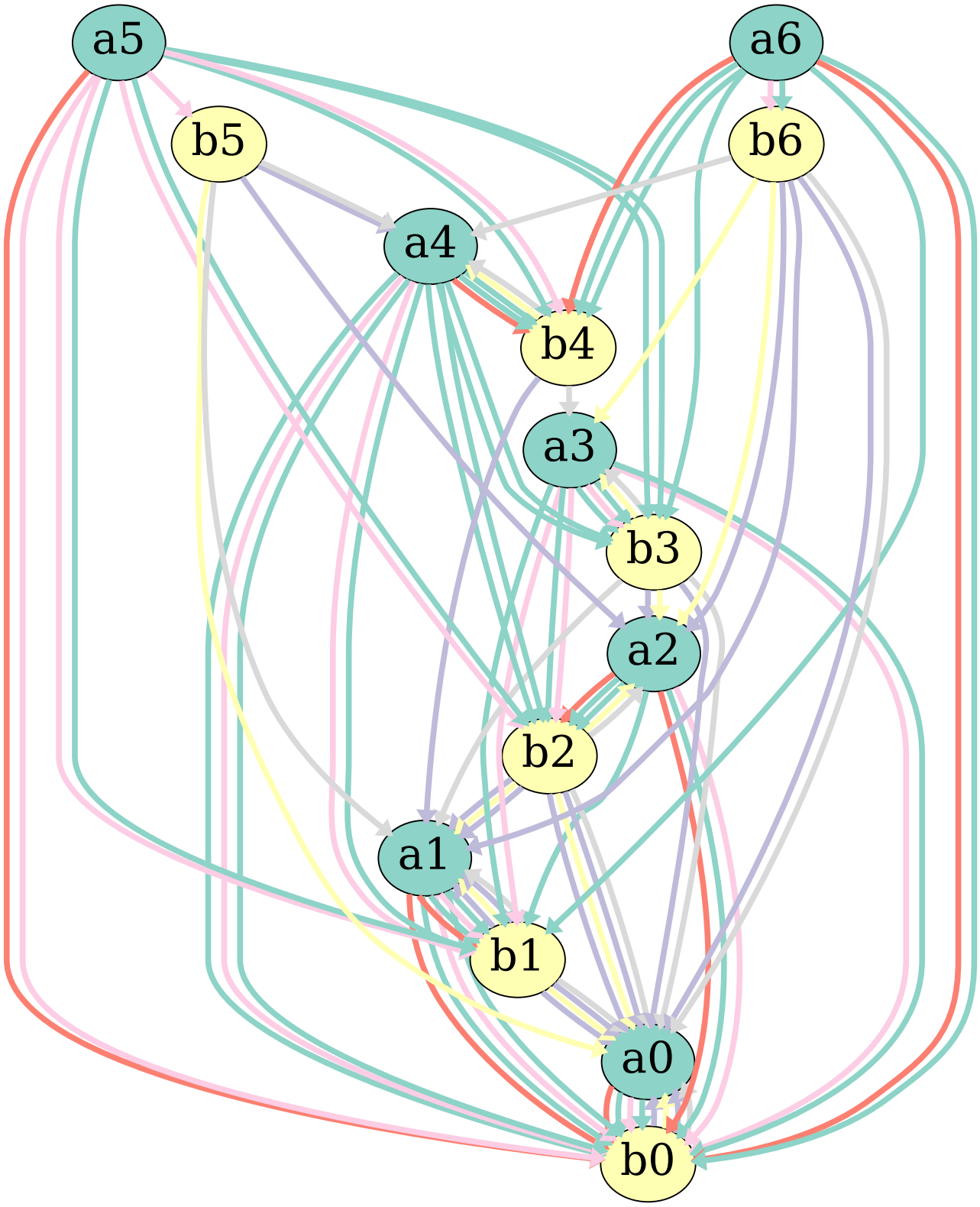


Figure 5: The searched architecture with $K_{Lo} = 16$ on Ecomm dataset, which achieves performance of 84.2% AUC, while is nearly 44% faster in inference time than the searched architecture with $K_{Lo} = 64$.