New Insights in Neural Architecture Search: From Architecture Embeddings to Learning Curve Extrapolations

Shen Yan
Michigan State University

Dec 3, 2021

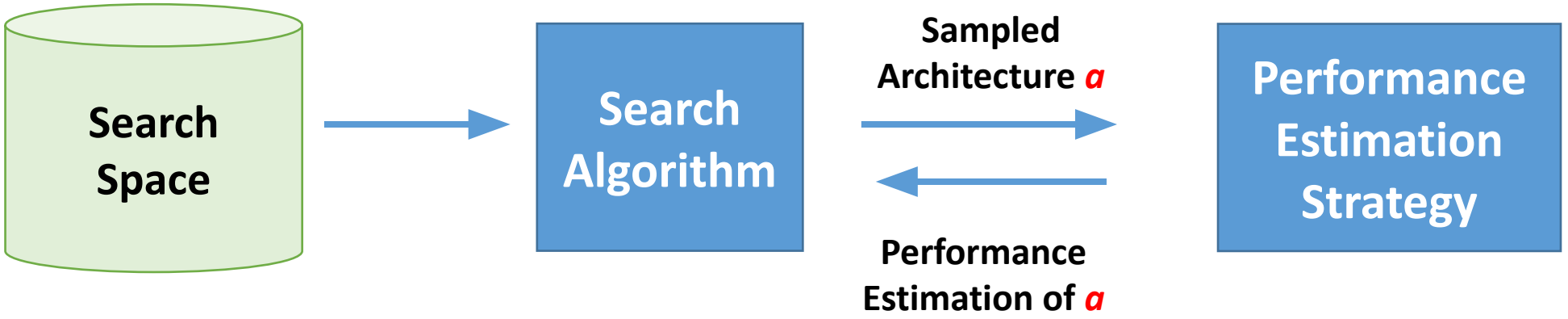
Outline

- **Background of NAS**
  - Search Space and Architecture Encodings
  - Search Algorithms and Performance Estimations

- **A Study on Neural Architecture Representations**
  - Why Decouples Architecture Representations and Search Algorithms
  - Visualizations and Analysis
  - From Encoding Structures of Neural Networks to Computations

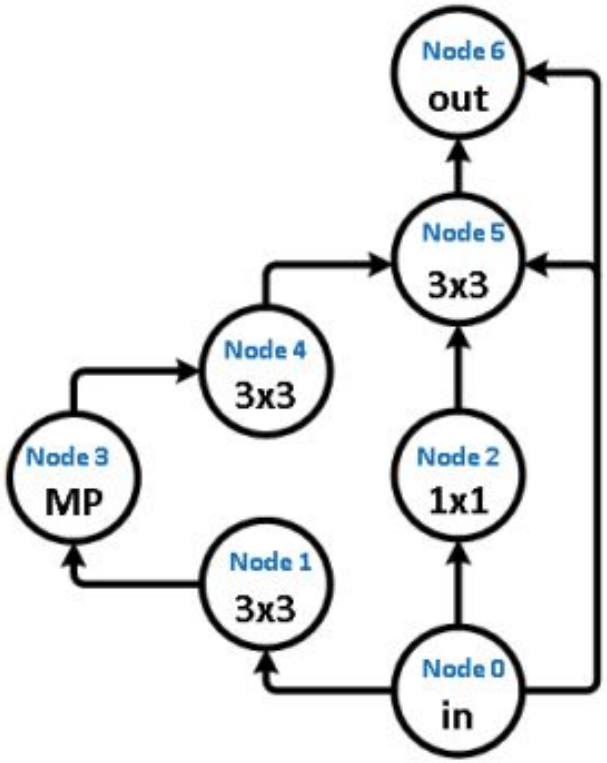
- **Speedy Performance Estimation via Learning Curve Extrapolation**
  - Create a Multi-Fidelity Surrogate Benchmark for Evaluating Anytime NAS algorithms
  - Speedy Estimations via Partial Curve Extrapolations

Neural Architecture Search (NAS) Pipeline



Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, Neural Architecture Search: A Survey, JMLR 2019

Search Space

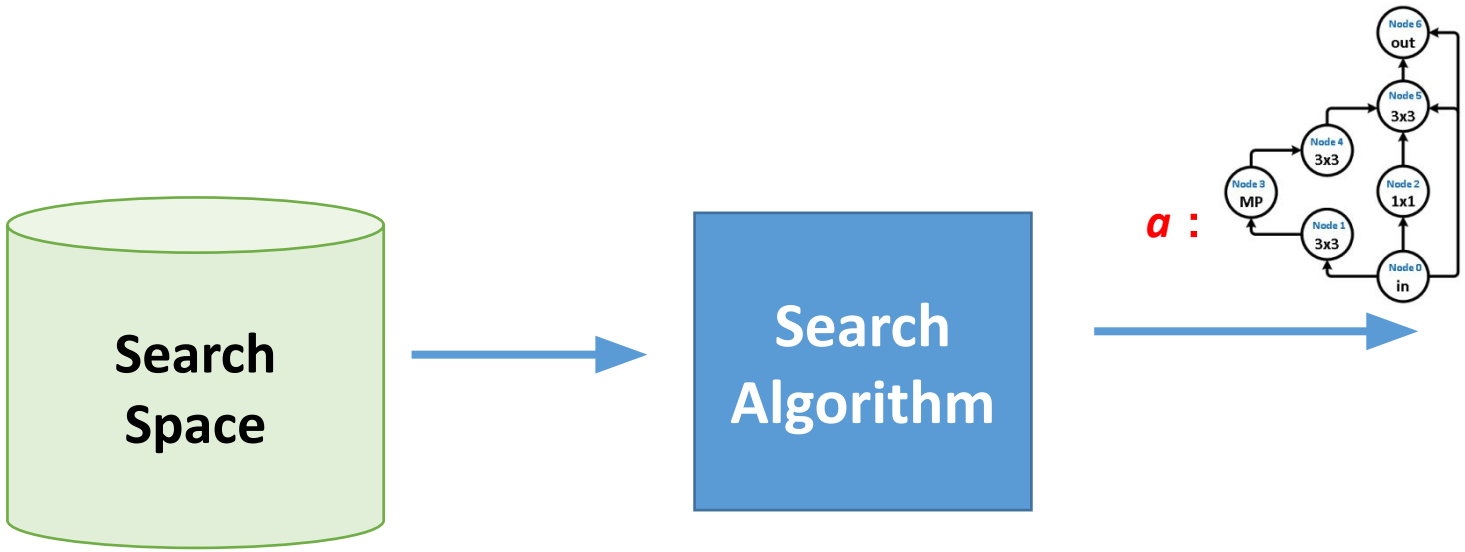


Set of Operations:

- Identity
- 3x3 convolution
- 1x1 convolution
- 3x3 max pooling
- skip connection

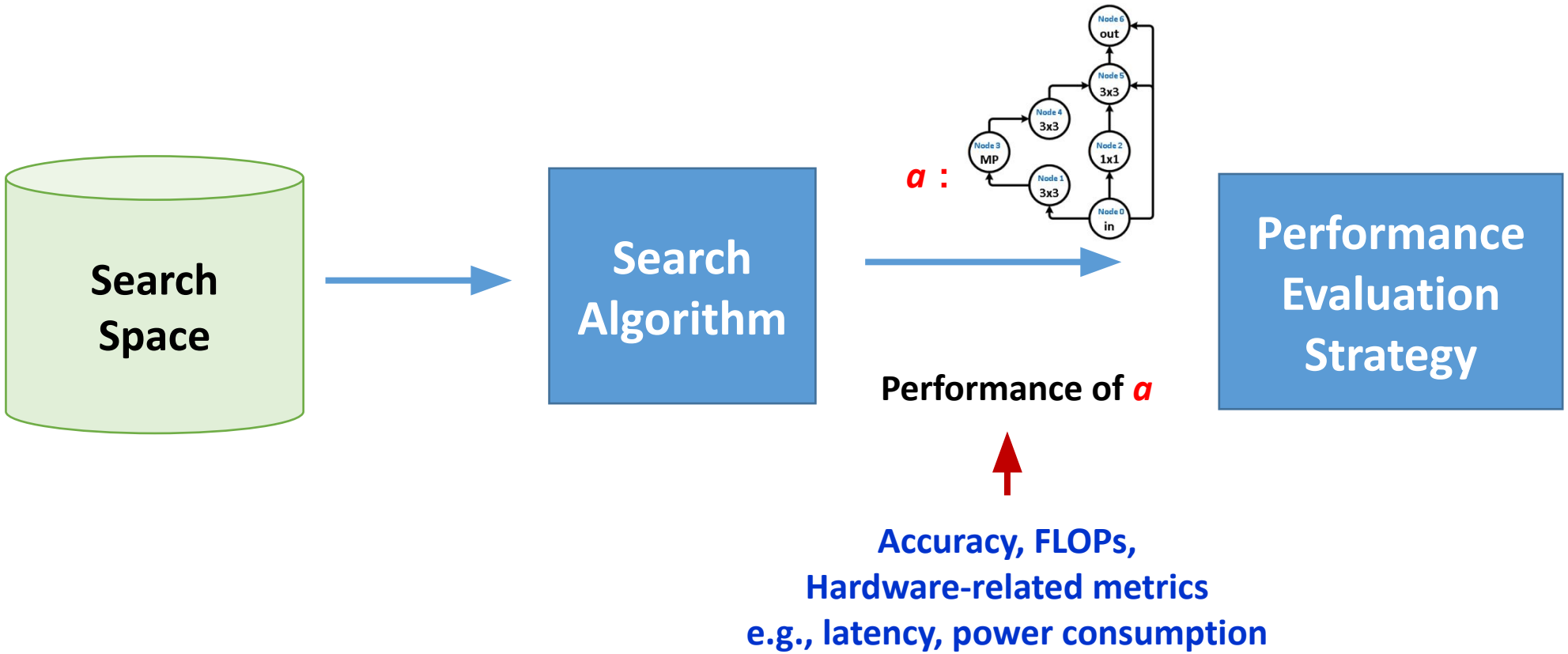
Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, Frank Hutter., NAS-Bench-101: Towards Reproducible Neural Architecture Search, JMLR 2019

Search Algorithm + Performance Evaluation

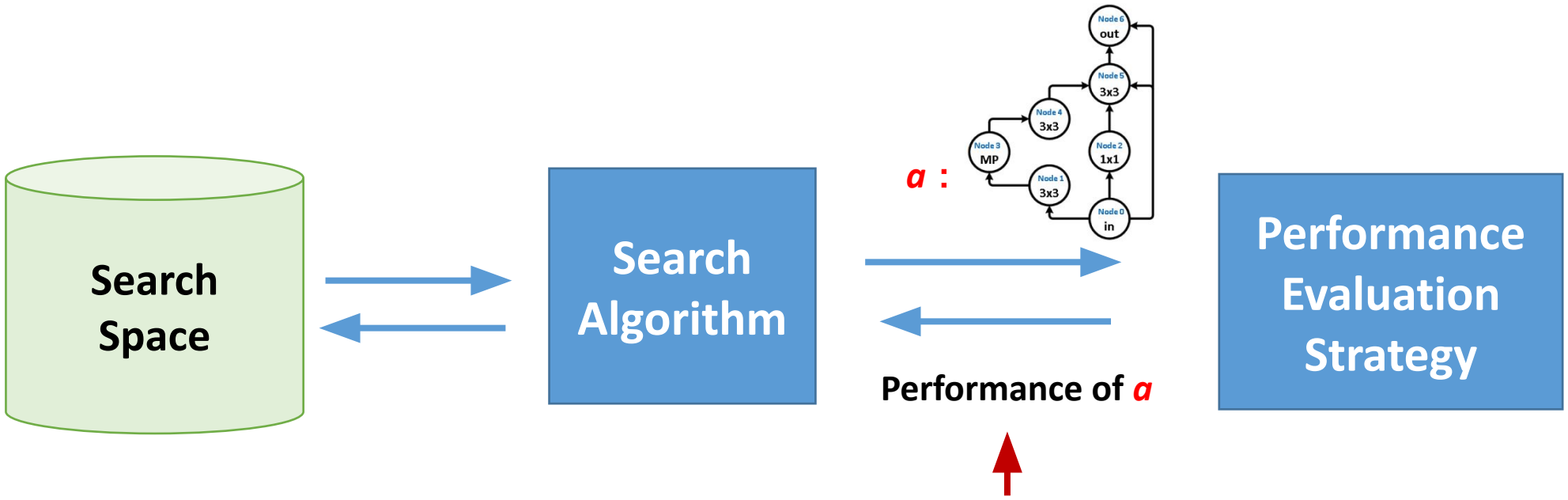


e.g. Random Search, Evolution, Reinforcement Learning, Bayesian Optimization, One-shot (supernet)

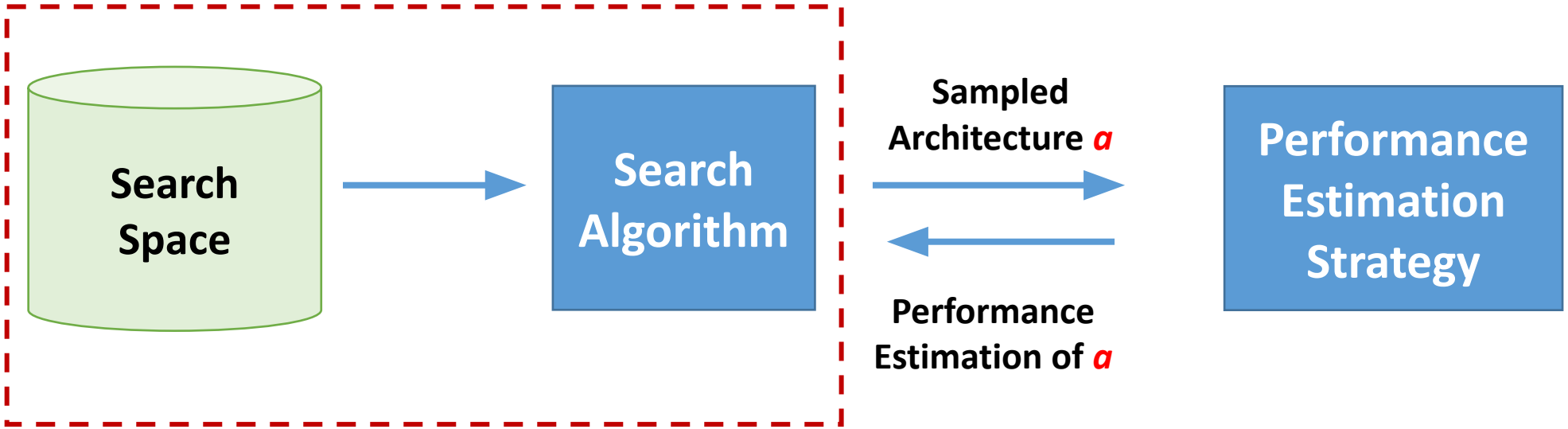
Search Algorithm + Performance Evaluation



Search Algorithm + Performance Evaluation

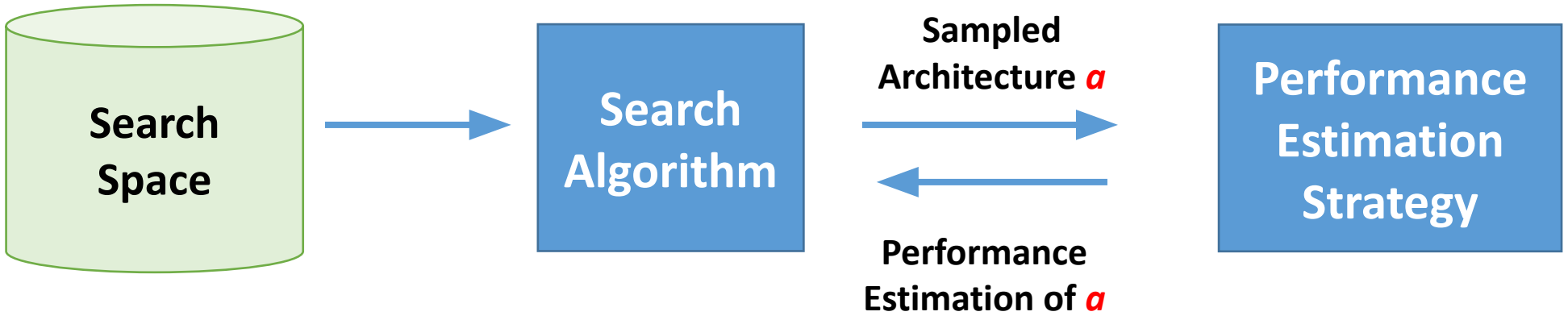


Recall this Figure



- Supernet-based training optimizes search space and search algorithm together
- The optimization of the architecture representations are guided by the accuracies of architectures selected by the search strategies
- What if the accuracies are bad at the beginning (e.g. cool-start)?

From the Classic NAS Framework



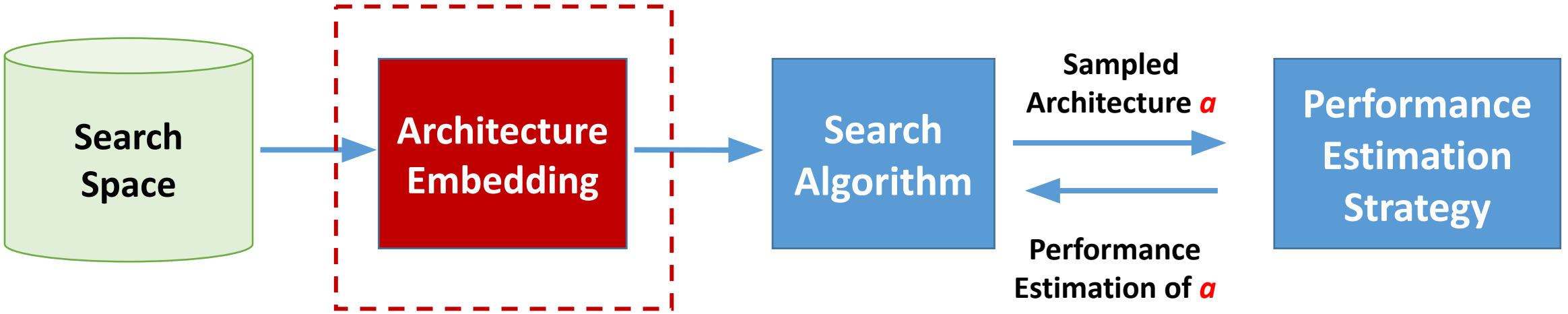
Search Space Design and Partitioning

Evolution (EA), Reinforcement Learning (RL), Bayesian Optimization (BO), Gradient Descent (GD), Neural Predictor (NP)

Weight Sharing, Lower Fidelity Estimates, Bandit-based, Learning Curve Extrapolation

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, Neural Architecture Search: A Survey, JMLR 2019

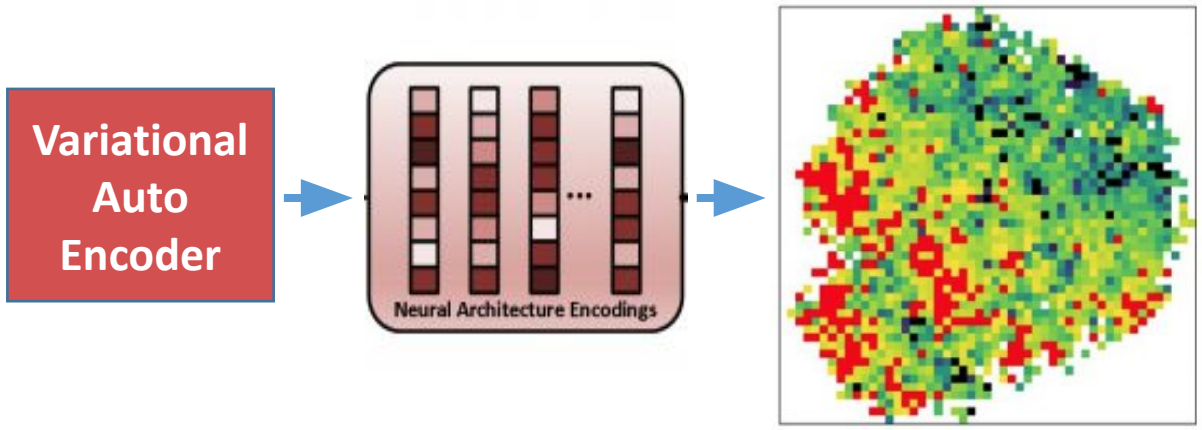
To a Study on Architecture Embeddings



In the following, I discuss about our research on decoupling neural network representation learning and its downstream search, as well as the effect of architecture representations on the overall performance of NAS.

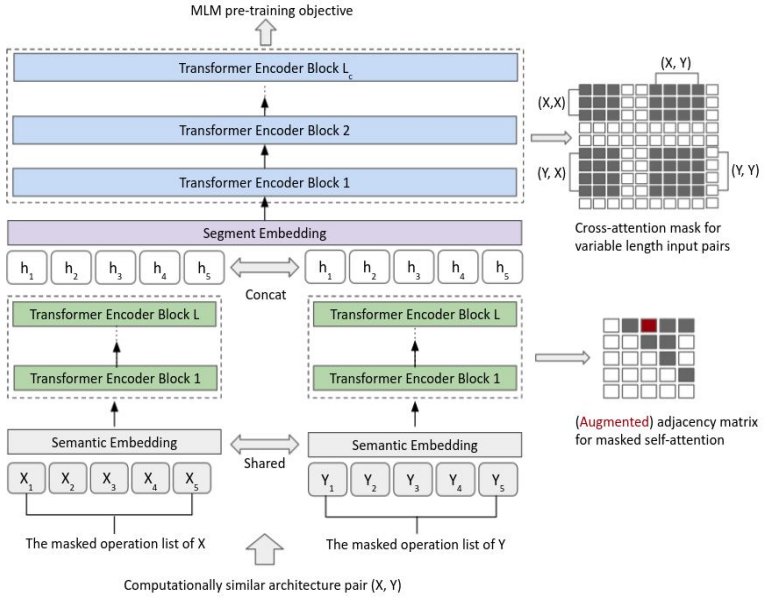
arch2vec and CATE

arch2vec [NeurIPS' 20]



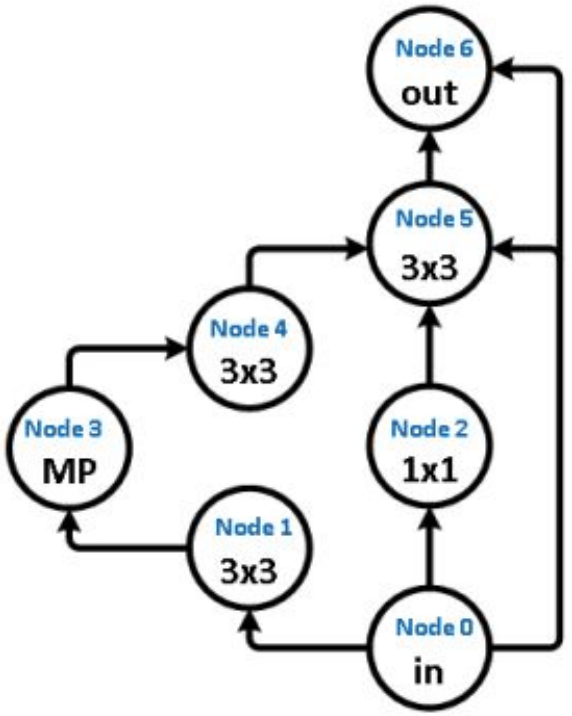
arch2vec is a structure-aware unsupervised learning-based architecture encoding method that **decouples** architecture representation learning and architecture search into two separate processes.

CATE [ICML' 21]



CATE is a computation-aware architecture encoding method that encodes **computations** instead of structures of neural architectures via a transformer-based encoder.

Recap: Adjacency Matrix-based Encoding



Encoding

Node

Node	[0	1	2	3	4	5	6]
0	0	1	1	0	0	1	1
1	0	0	0	1	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	0	1	0	0
4	0	0	0	0	0	1	0
5	0	0	0	0	0	0	1
6]	0	0	0	0	0	0	0

Adjacency Matrix

Encode connections between operations

Operation

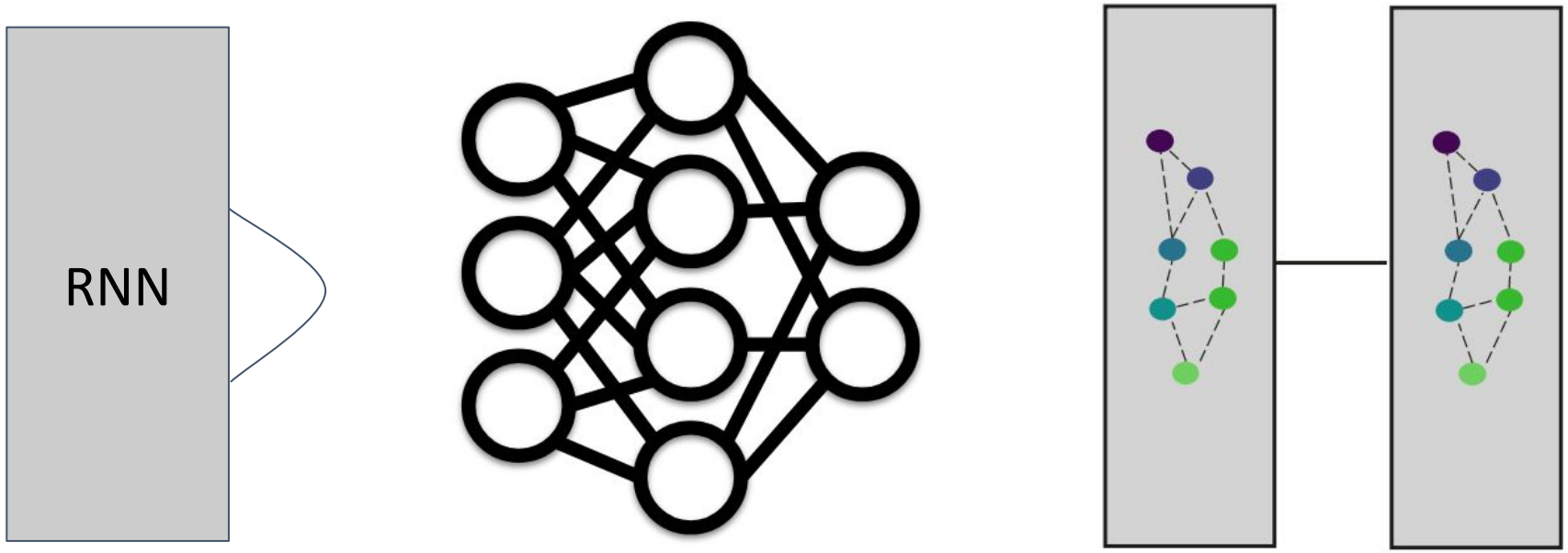
Node	[in	1x1	3x3	MP	out]
0	1	0	0	0	0
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	0	1	0	0
5	0	0	1	0	0
6]	0	0	0	0	1

Operation Matrix

Encode operations

The size of the adjacency matrix grows **quadratically** as search space scales up, making the downstream architecture search **much less efficient**.

Recap: Learning-based Embeddings



Different Types of Architecture Encoders such as LSTM, MLP, and GCN

Why Learning-based Embeddings

Adjacency Matrix-based Encoding

Node	[0	1	2	3	4	5	6]
0	0	1	1	0	0	1	1
1	0	0	0	1	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	0	1	0	0
4	0	0	0	0	0	1	0
5	0	0	0	0	0	0	1
6]	0	0	0	0	0	0	0

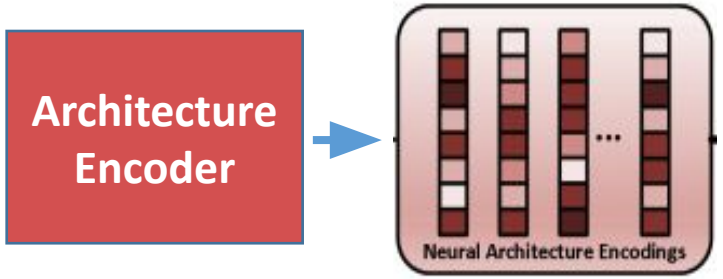
Adjacency Matrix

Node	[in	1x1	3x3	MP	out]
0	1	0	0	0	0
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	0	1	0	0
5	0	0	1	0	0
6]	0	0	0	0	1

Operation Matrix

High-dimensional & discrete search space

Learning-based Embeddings

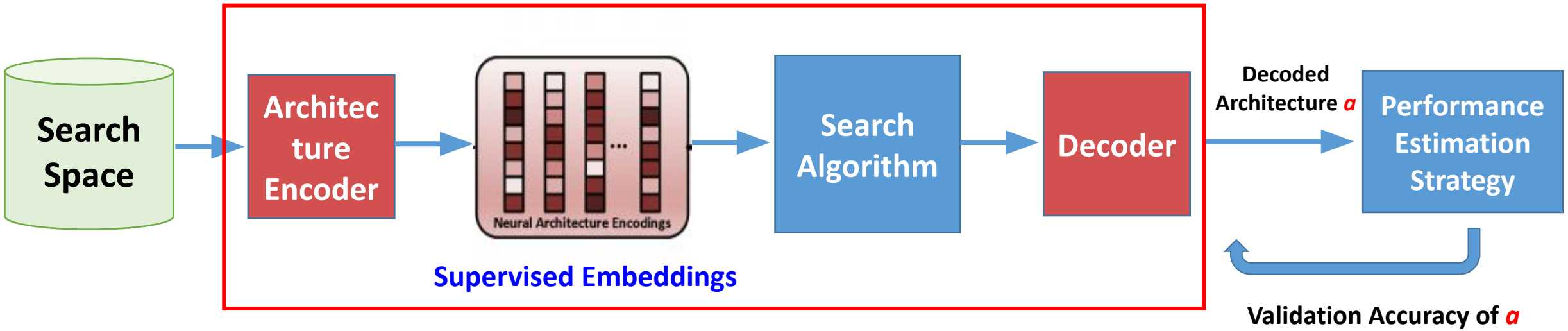


Learned architecture embeddings in the Low-dimensional latent space

Conducting neural architecture search on such low-dimensional continuous space is much easier and is hence more efficient.

Drawback of Existing Learning-based Embedding Methods

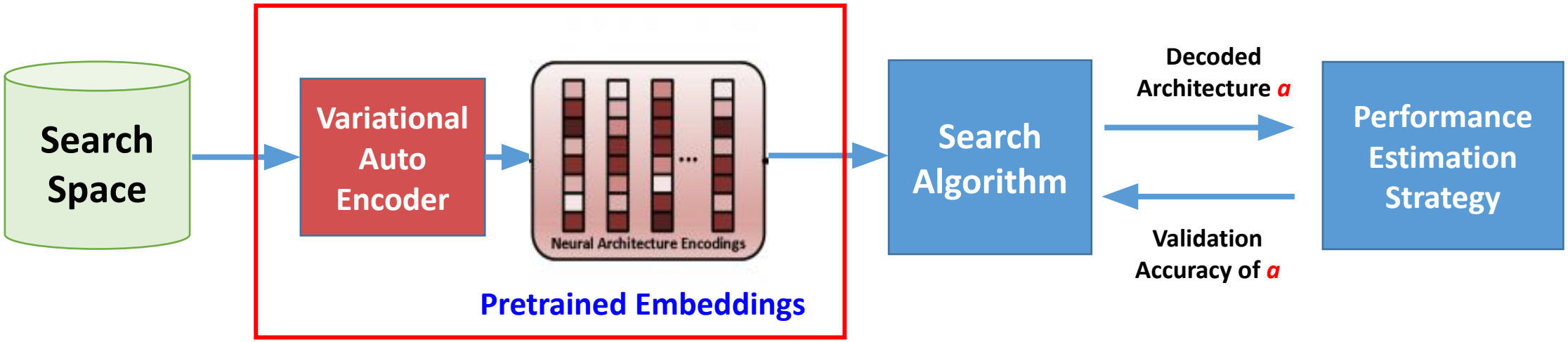
- Architecture embeddings and search algorithms are ***jointly*** optimized in a ***supervised*** manner, guided by the accuracies of architectures selected by the search strategies.



Architecture embeddings and search strategies are jointly optimized in a supervised manner

Our Proposed Method: arch2vec

- We propose **arch2vec**, a simple yet effective ***unsupervised*** architecture representation learning method for neural architecture search.
- ***Decouple*** architecture embedding learning and architecture search into two ***separate*** processes.



Pre-training architecture embeddings in an unsupervised manner

Variational Graph Isomorphism Autoencoder

Let **A** denote **Adjacency Matrix**, **X** denote **Operation Matrix**.

Augment **A** as $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{A}^{\boldsymbol{T}}$ to transfer original directed graph into undirected one to allow bi-directional information flow.

Encoder

$$q(\mathbf{Z}|\mathbf{X}, \tilde{\mathbf{A}}) = \prod_{i=1}^{N} q(\mathbf{z}_i|\mathbf{X}, \tilde{\mathbf{A}}), \text{with } q(\mathbf{z}_i|\mathbf{X}, \tilde{\mathbf{A}}) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_i, diag(\boldsymbol{\sigma}_i^2)),$$

$$\mathbf{H}^{(k)} = \text{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot \mathbf{H}^{(k-1)} + \tilde{\mathbf{A}}\mathbf{H}^{(k-1)}\right), k = 1, 2, \ldots, L$$

L-layer Graph Isomorphism Network (GIN)

Decoder

$$p(\hat{\mathbf{A}}|\mathbf{Z}) = \prod_{i=1}^{N}\prod_{j=1}^{N} P(\hat{A}_{ij}|\mathbf{z}_i, \mathbf{z}_j), \text{with } p(\hat{A}_{ij} = 1|\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$$

Reconstructed Adjacency Matrix

$$p(\hat{\mathbf{X}} = [k_1, ..., k_N]^T|\mathbf{Z}) = \prod_{i=1}^{N} P(\hat{\mathbf{X}}_i = k_i|\mathbf{z}_i) = \prod_{i=1}^{N} \text{softmax}(\mathbf{W}_o\mathbf{Z} + \mathbf{b}_o)_{i,k_i}$$

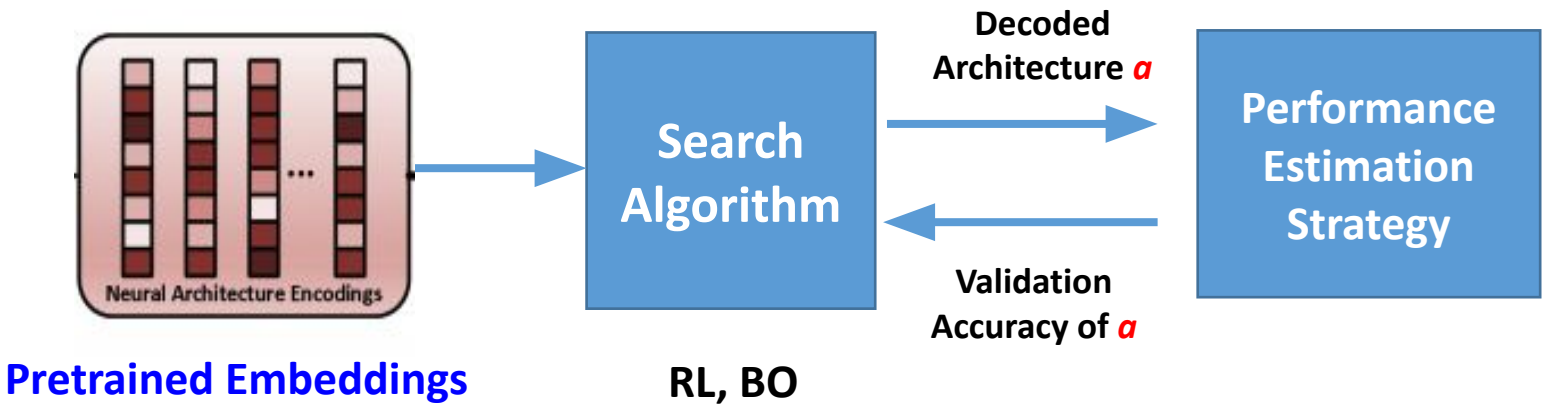
Reconstructed Operation Matrix

Training objective

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\tilde{\mathbf{A}})}[\log p(\hat{\mathbf{X}}, \hat{\mathbf{A}}|\mathbf{Z})] - \mathcal{D}_{KL}(q(\mathbf{Z}|\mathbf{X}, \tilde{\mathbf{A}})||p(\mathbf{Z}))$$

Pretrained Embeddings for Architecture Search

We use reinforcement learning (RL) and Bayesian optimization (BO) as two representative search algorithms.



- **RL**: State: pretrained embeddings. Policy Net: LSTM. Reward: Validation accuracy
- **BO**: Surrogate: 2-Layer MLP (instead of GP). Acquisition function: EI.
- Output top-K architectures in each round of search and optimize the policy/surrogate iteratively

Pre-training Performance

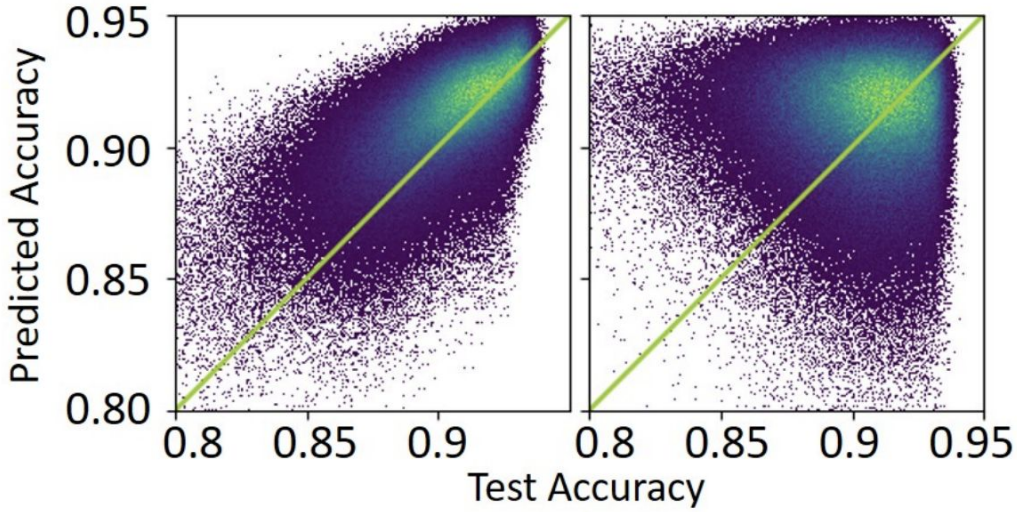
- Three commonly used NAS search spaces: **NAS-Bench-101**, **NAS-Bench-201**, and the **DARTS** search space.
- We compare *arch2vec* with two baselines: Graph Autoencoders (GAE) and Variational Graph Autoencoders (VGAE) under three metrics:
  - Reconstruction Accuracy: how accurate the reconstructed network architectures are.
  - Validity: how often the generated architectures are valid graphs.
  - Uniqueness: how many generated valid architectures are unique.

Method	NAS-Bench-101			NAS-Bench-201			DARTS		
	Accuracy	Validity	Uniqueness	Accuracy	Validity	Uniqueness	Accuracy	Validity	Uniqueness
GAE [27]	98.75	29.88	99.25	99.52	79.28	78.42	97.80	15.25	99.65
VGAE [27]	97.45	41.18	99.34	98.32	79.30	88.42	96.80	25.25	99.27
arch2vec	**100**	**51.33**	**99.36**	**100**	**79.41**	**98.72**	**99.79**	**33.36**	**100**

Understanding Pre-trained Embeddings (1)

- We compare the predictive performance of the pretrained embeddings and supervised embeddings. This metric measures how well the embeddings can predict the performance of the corresponding architectures.

- We train a Gaussian Process model with 250 sampled data to predict all data and report the results across 10 different seeds. We use RMSE and the Pearson correlation coefficient to evaluate points with test accuracy larger than 0.8.



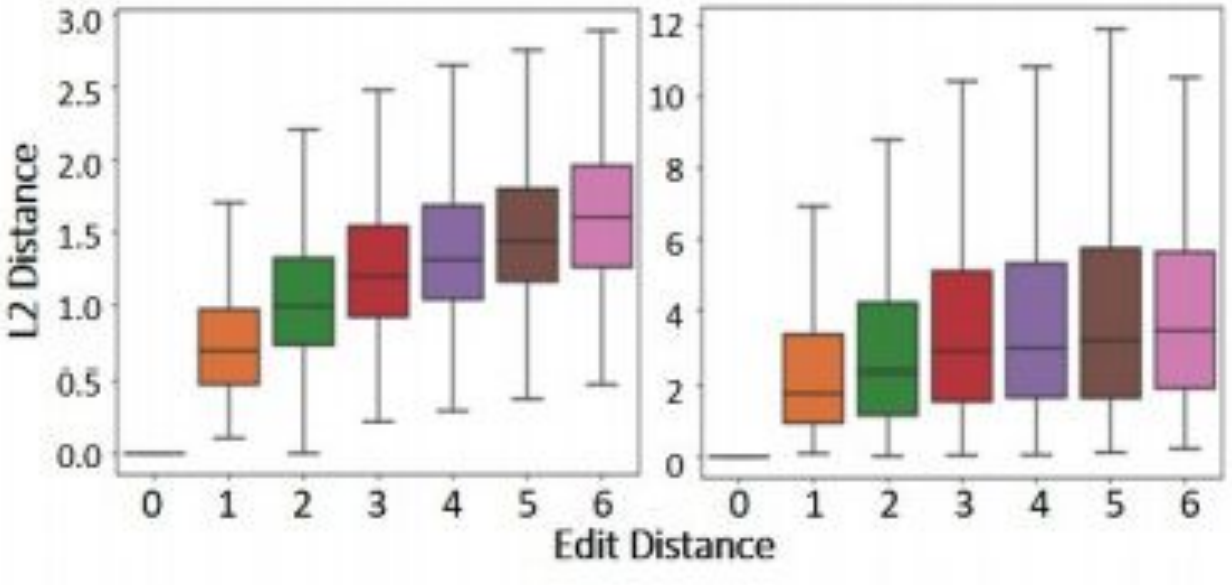
Pretrained Embeddings (arch2vec)

Supervised Embeddings

The RMSE and Pearson's r are: 0.038±0.025 / 0.53±0.09 for supervised embeddings, and 0.018±0.001 / 0.67 ±0.02 for *arch2vec*.

Understanding Pre-trained Embeddings (2)

- We compare the distribution of L2 distance between architecture pairs by edit distance, measured by 1,000 architectures sampled in a long random walk with 1 edit distance apart from consecutive samples.
- The L2 distance of pretrained embeddings grows monotonically with increasing edit distance.
- This observation indicates that the pretrained embeddings are able to better capture the structural information of neural networks, and thus make similar architectures clustered better.

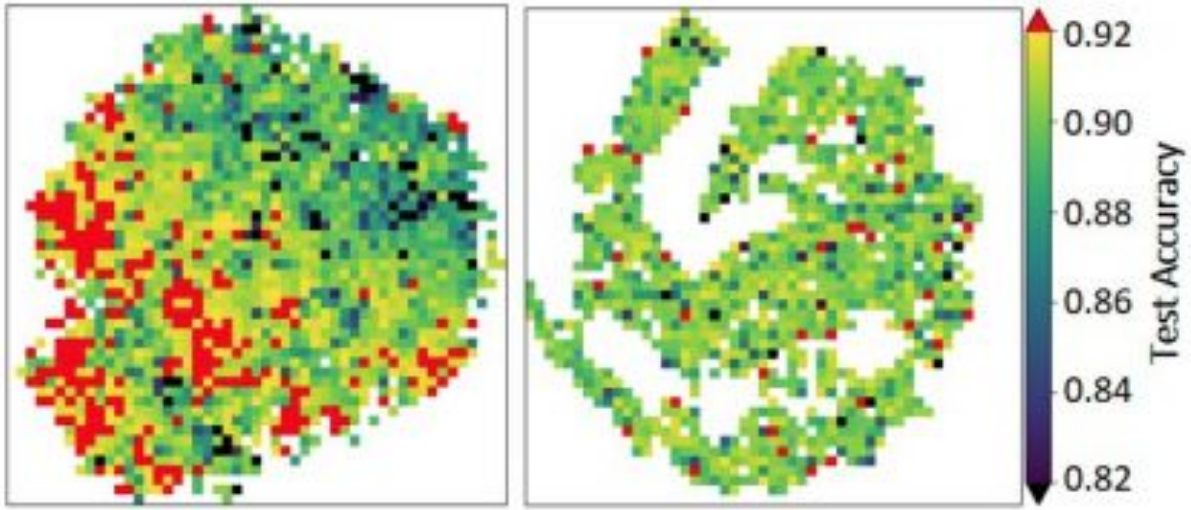


Pretrained Embeddings (arch2vec)

Supervised Embeddings

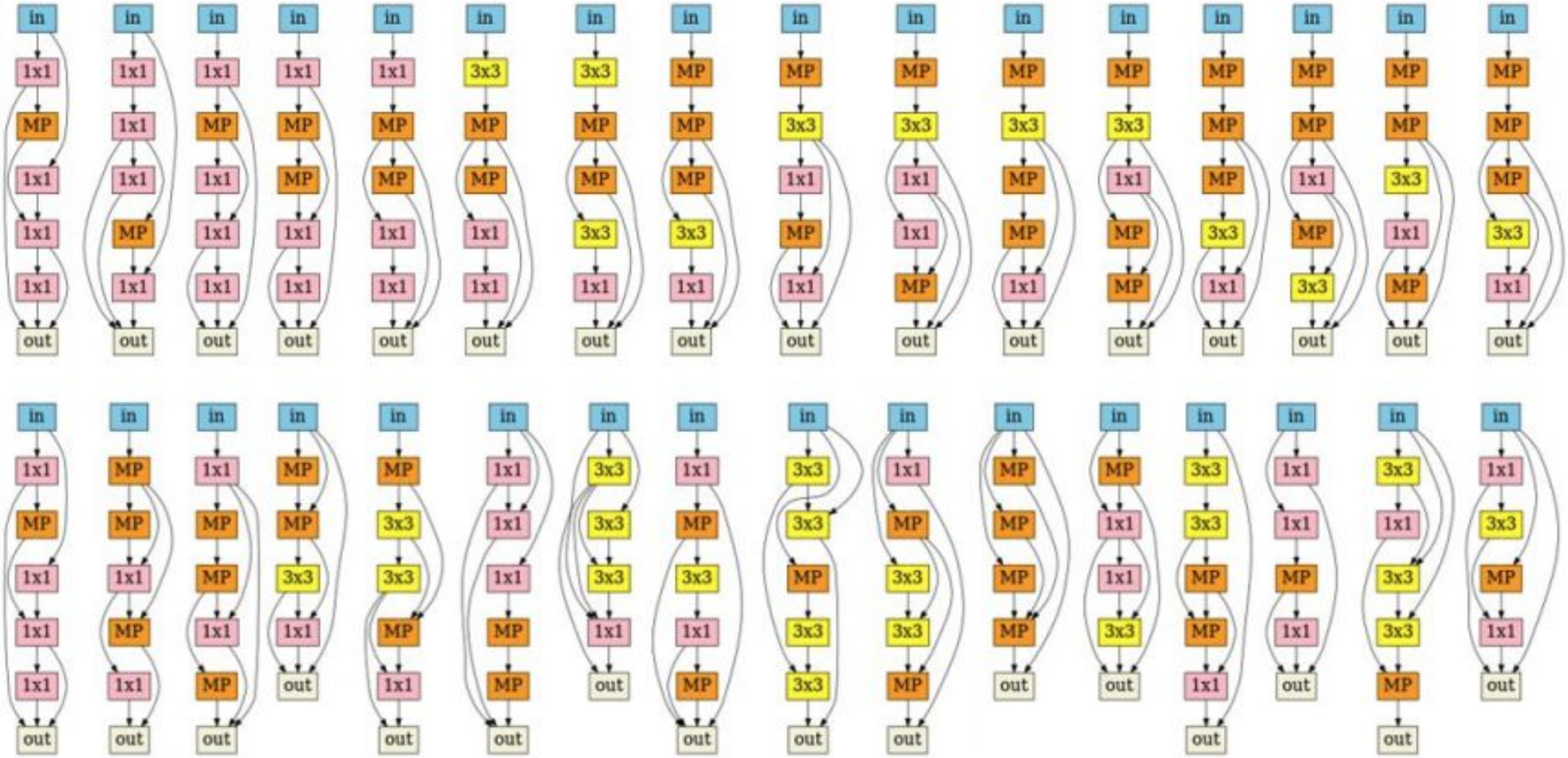
Understanding Pre-trained Embeddings (3)

- We visualize the latent spaces learned by *arch2vec* and its supervised learning counterpart in 2-dimensional space.
- Compared to supervised embeddings, pretrained embeddings span the whole latent space, and architectures with similar accuracies are clustered and distributed more smoothly in the latent space.
- Conducting architecture search on such smooth performance surface is much easier and is hence more efficient.



Latent space 2D visualization comparison between arch2vec (left) and supervised architecture representation learning (right). Color encodes test accuracy.

Understanding Pre-trained Embeddings (4)



Pretrained Embeddings
(edit distances between adjacent architectures are 4, 6, 1, 5, 1, 1, 1, 5, 2, 3, 2, 4, 2, 5, 2;)

Supervised Embeddings
(edit distances between adjacent architectures are 8, 6, 7, 7, 9, 8, 11, 11, 6, 10, 10, 11, 10, 11, 9)

Understanding Pre-trained Embeddings (4)

- **The locality around the global maximum as well and the peak also has a coarse-grained width of about 6**
  - The RWA is nearly 0 during the joint optimization

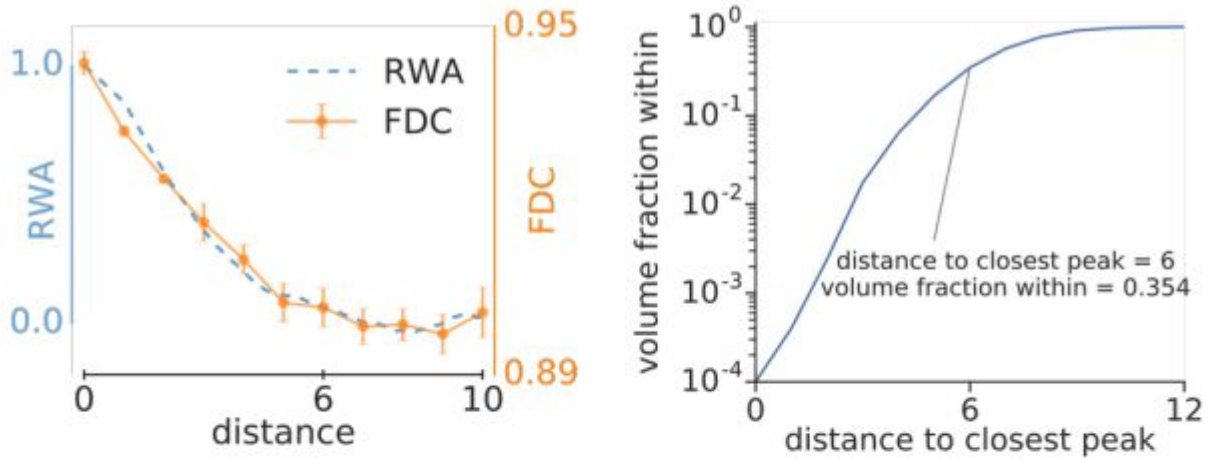
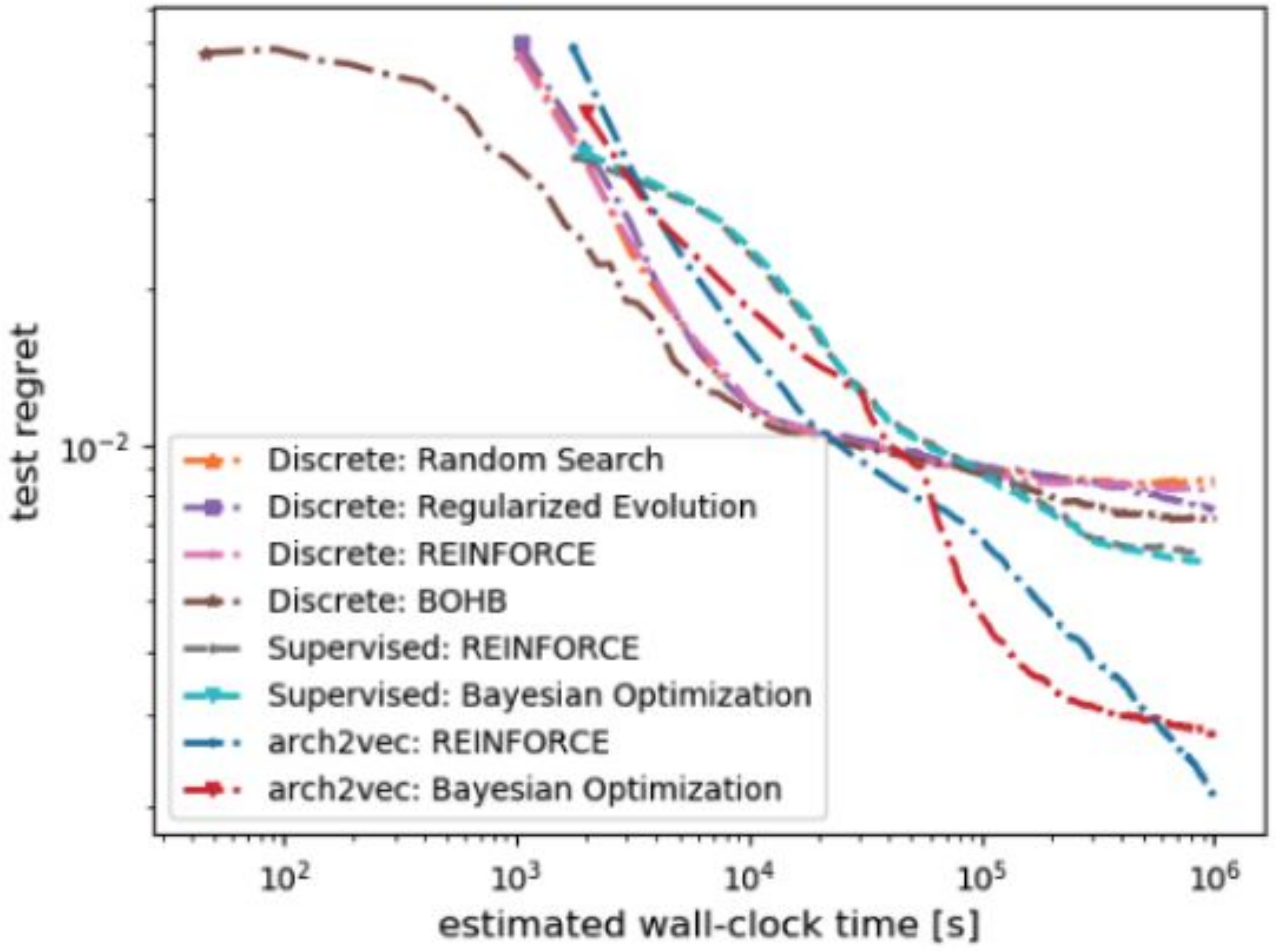


Figure 6: (left) RWA for the full space and the FDC relative to the global maximum. To plot both curves on a common horizontal axis, the autocorrelation curve is drawn as a function of the square root of the autocorrelation shift, to account for the fact that a random walk reaches a mean distance $\sqrt{N}$ after N steps. (right) Fraction of the search space volume that lies within a given distance to the closest high peak.

NAS-Bench-101: Towards Reproducible Neural Architecture Search. Ying et al. ICML 2019

Architecture Search Performance on NAS-Bench-101

- BOHB and RE are two best-performing search methods using discrete encoding.
- However, they perform slightly worse than supervised architecture representation learning.
- *arch2vec* considerably outperforms its supervised counterpart and the discrete encoding after 50,000 wall clock seconds.



Architecture Search Performance on NAS-Bench-201

- Searching with *arch2vec* consistently outperforms other approaches on all the three datasets in NAS-Bench-201, leading to better validation and test accuracy as well as reduced variability.

NAS Methods	CIFAR-10		CIFAR-100		ImageNet-16-120	
	validation	test	validation	test	validation	test
RE [41]	91.08±0.43	93.84±0.43	73.02±0.46	72.86±0.55	45.78±0.56	45.63±0.64
RS [59]	90.94±0.38	93.75±0.37	72.17±0.64	72.05±0.77	45.47±0.65	45.33±0.79
REINFORCE [10]	91.03±0.33	93.82±0.31	72.35±0.63	72.13±0.79	45.58±0.62	45.30±0.86
BOHB [12]	90.82±0.53	93.61±0.52	72.59±0.82	72.37±0.90	45.44±0.70	45.26±0.83
arch2vec-RL	91.32±0.42	94.12±0.42	73.13±0.72	73.15±0.78	46.22±0.30	46.16±0.38
arch2vec-BO	**91.41±0.22**	**94.18±0.24**	**73.35±0.32**	**73.37±0.30**	**46.34±0.18**	**46.27±0.37**

Architecture Search Performance on DARTS

	Test Error		Params (M)	Search Cost				
NAS Methods	Avg	Best		Stage 1	Stage 2	Total	**Encoding**	**Search Method**
Random Search [15]	3.29±0.15	-	3.2	-	-	4	-	Random
ENAS [68]	-	2.89	4.6	0.5	-	-	Supervised	REINFORCE
ASHA [69]	3.03±0.13	2.85	2.2	-	-	9	-	Random
RS WS [69]	2.85±0.08	2.71	4.3	2.7	6	8.7	-	Random
SNAS [16]	2.85±0.02	-	2.8	1.5	-	-	Supervised	GD
DARTS [15]	2.76±0.09	-	3.3	4	1	5	Supervised	GD
BANANAS [49]	2.64	2.57	3.6	100 (queries)	-	11.8	Supervised	BO
Random Search (ours)	3.1±0.18	2.71	3.2	-	-	4	-	Random
DARTS (ours)	2.71±0.08	2.63	3.3	4	1.2	5.2	Supervised	GD
BANANAS (ours)	2.67±0.07	2.61	3.6	100 (queries)	1.3	11.5	Supervised	BO
arch2vec-RL	2.65±0.05	2.60	3.3	100 (queries)	1.2	9.5	Unsupervised	REINFORCE
arch2vec-BO	**2.56±0.05**	**2.48**	3.6	100 (queries)	1.3	10.5	Unsupervised	BO

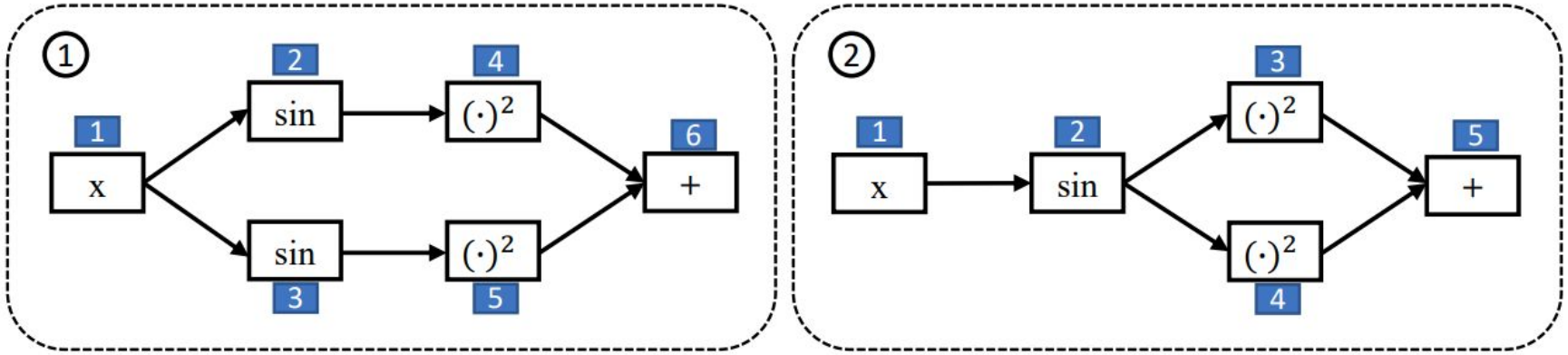
Table 4: Comparison with state-of-the-art cell-based NAS methods on DARTS search space using CIFAR-10. The test error is averaged over 5 seeds. Stage 1 shows the GPU days (or number of queries) for model search and Stage 2 shows the GPU days for model evaluation.

NAS Methods	**Params (M)**	**Mult-Adds (M)**	**Top-1 Test Error (%)**	**Comparable Search Space**
NASNet-A [62]	5.3	564	26.0	Y
AmoebaNet-A [44]	5.1	555	25.5	Y
PNAS [44]	5.1	588	25.8	Y
SNAS [16]	4.3	522	27.3	Y
DARTS [15]	4.7	574	26.7	Y
arch2vec-RL	4.8	533	25.8	Y
arch2vec-BO	5.2	580	25.5	Y

Table 5: Transfer learning results on ImageNet.

Drawback of Structure-aware Encoding

Two neural networks below with different structures represent the same computation: $\sin(x)^2 + \sin(x)^2$.

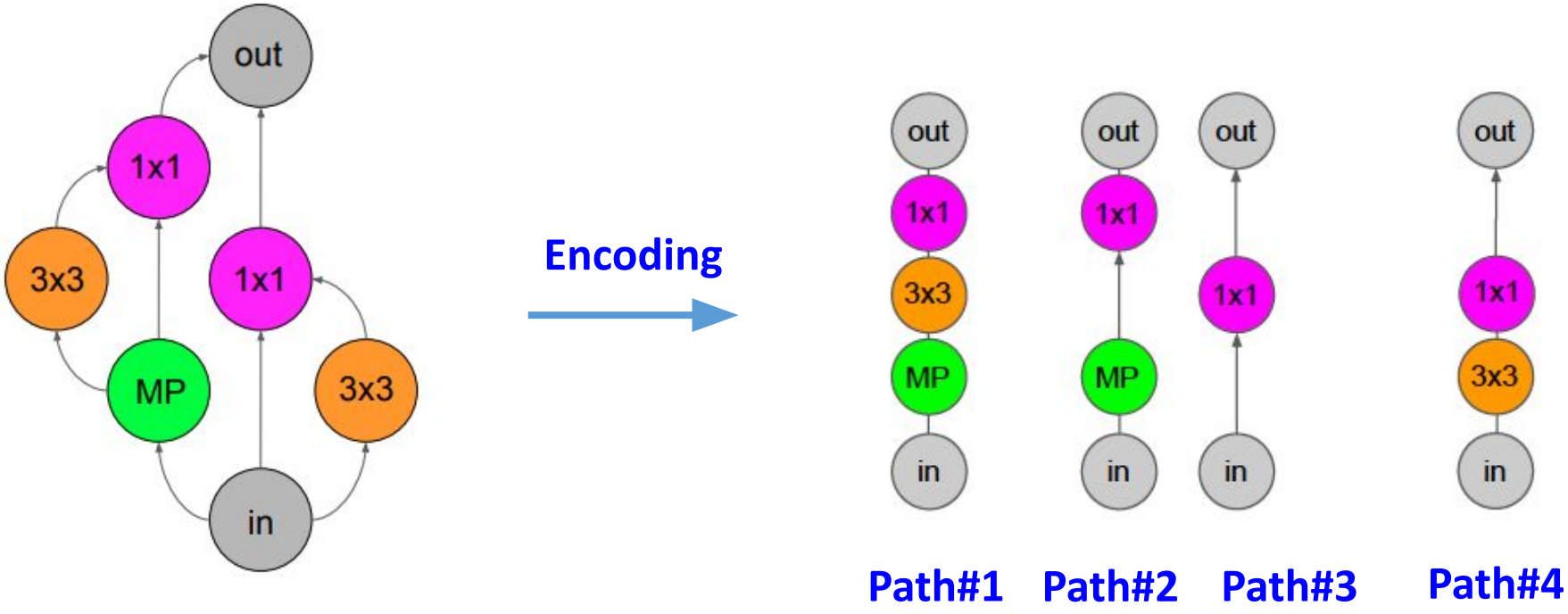


Encode structures of them will result in the same embedding located in different regions in the latent space.

Instead, encoding computation contributes to an encoding space with respect to the actual performance of the neural networks.

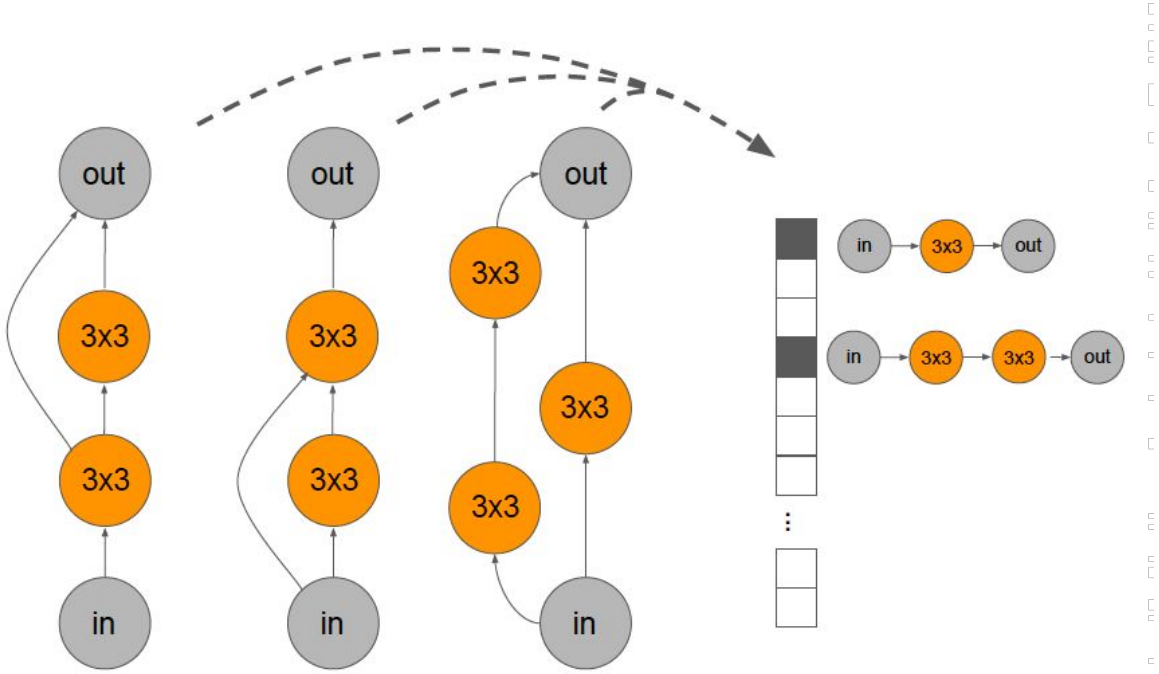
Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, Yixin Chen., D-VAE: A Variational Autoencoder for Directed Acyclic Graphs, NeurIPS 2019

Computation-aware Encoding: Path-based Encoding



Colin White, Willie Neiswanger, Sam Nolen, Yash Savani., A Study on Encodings for Neural Architecture Search, NeurIPS 2020

Computation-aware Encoding: Path-based Encoding



Pros

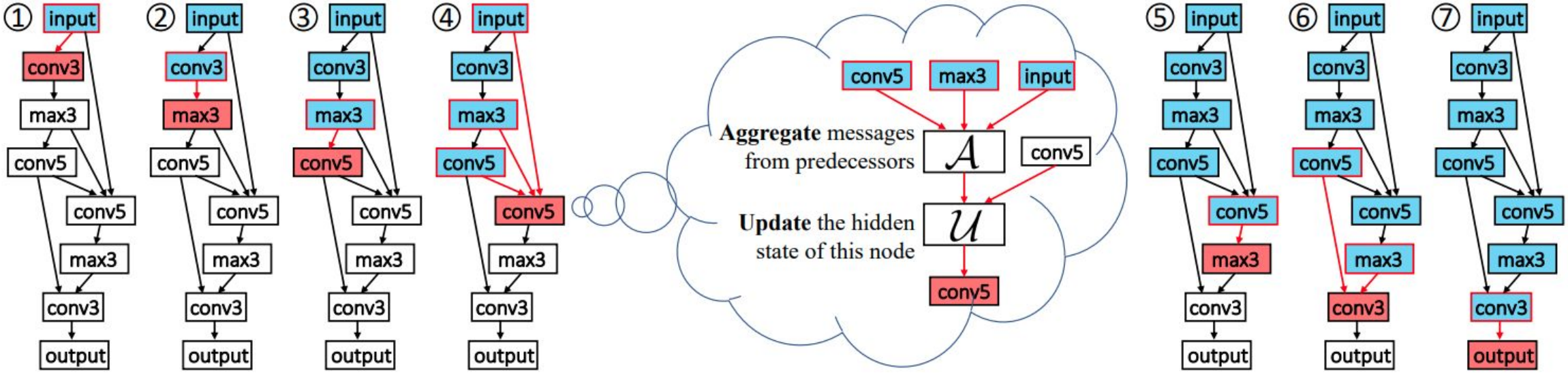
- Maps neural networks with different structures but the same computation to the same encoding.

Cons

- Scales exponentially without truncation but causes information loss with truncation.
- Shows worse generalization performance in outside search space compared to adjacency matrix-based encoding.

Colin White, Willie Neiswanger, Sam Nolen, Yash Savani., A Study on Encodings for Neural Architecture Search, NeurIPS 2020

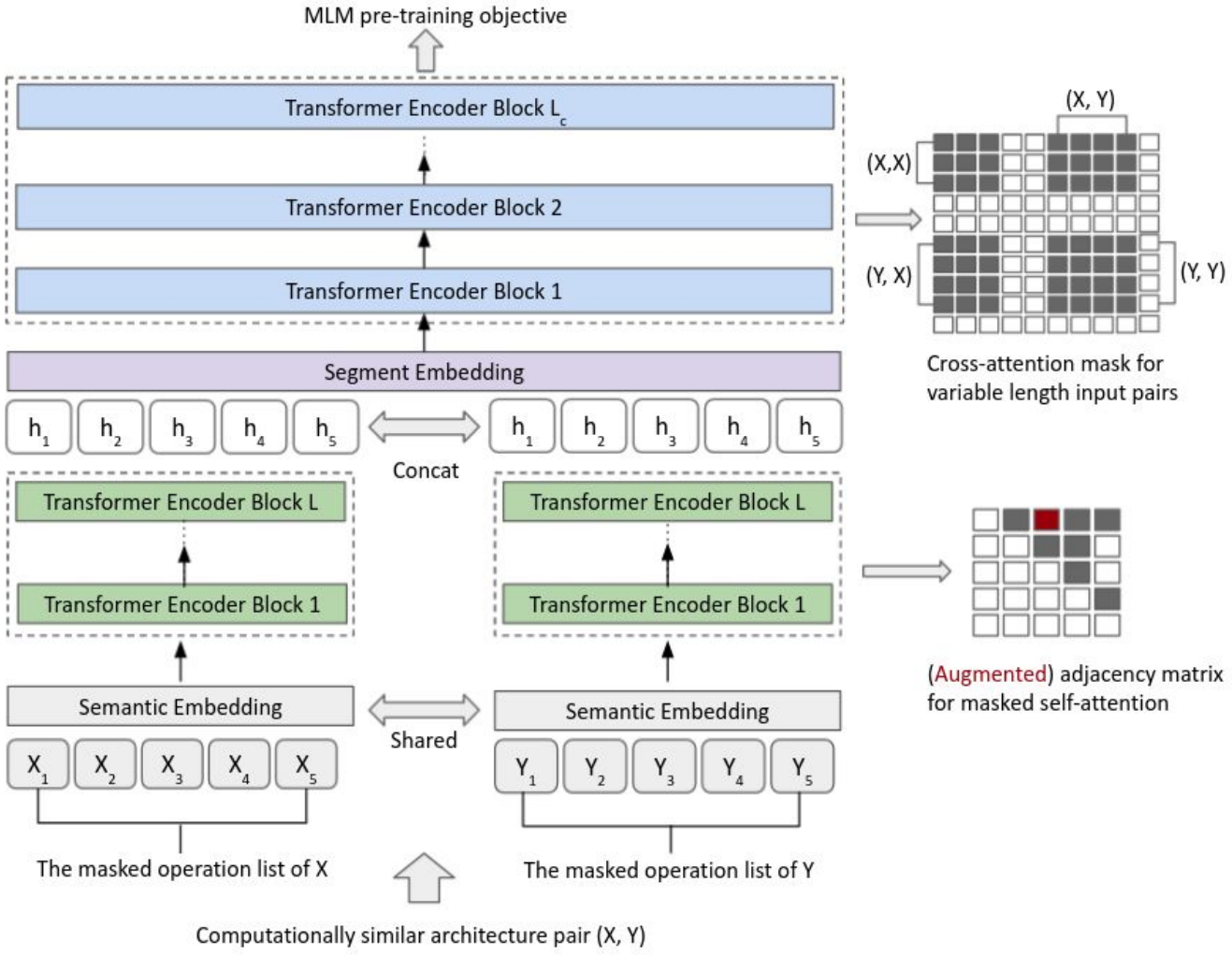
Computation-aware Encoding: D-VAE



Directly learning the generative model based on a single architecture is not trivial.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, Yixin Chen., D-VAE: A Variational Autoencoder for Directed Acyclic Graphs, NeurIPS 2019

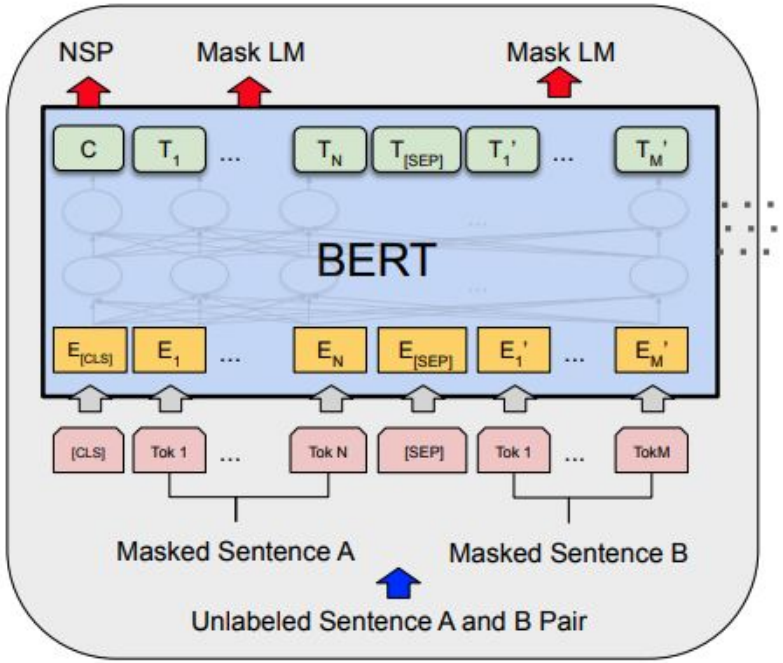
Our Proposed Method: CATE
(Computation-Aware Transformer-based Encoding)



- CATE takes **paired computationally similar architectures** as its input.
- Similar to BERT, CATE trains a **Transformer**-based model using the **masked autoencoding (MAE) objective**.
- Each input architecture pair is corrupted by replacing a fraction of their operators with a special [*MASK*] token.
- The model is trained to predict those masked operators from the corrupted architecture pair.

Why Pairwise Pre-training

Language Modeling



Architecture Encoding

Conv 3x3 -> Conv 1x1 -> Conv 5x5 -> ~~Conv 1x1~~ -> Max Pool -> ...

Conv 3x3 -> Conv 1x1 -> Conv 5x5 -> ? -> Max Pool -> ...

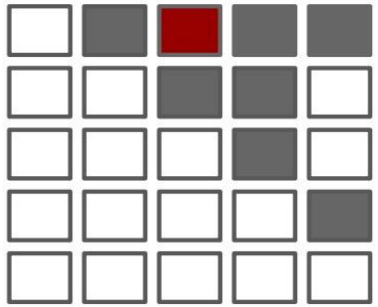
- Challenge: unlike LM, each prediction is **uniformly distributed**
- Solution: condition on **pairwise computationally similar architectures**

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, NAACL-HLT 2019

Offline Architecture Pair Sampling

- Sort the architectures based on their computational attributes P (e.g. number of parameters, FLOPs).
- Use a sliding window for each architecture x_i and its neighborhood $r(x_i) = \{y : |P(x_i) - P(y)| < \delta\}$, where δ is a hyperparameter for the pairwise computation constraint.
- Finally, we randomly select K distinct architectures $Y = \{y_1, \ldots, y_K\}$, x_i not $\in Y$, $Y \subset r(x_i)$ within the neighborhood to compose K architecture pairs $\{(x_i, y_1), \ldots, (x_i, y_K)\}$ for architecture x_i
- We use $\delta = 1e^6, 2e^6, 4e^6, 8e^6$, $K = 1, 2, 4, 8$ in our experiments.

Attention Mask



(Augmented) adjacency matrix for masked self-attention

Algorithm 1 Floyd Algorithm

Input: the node set $\mathcal{V}$, the adjacent matrix $\mathbf{A}$
$\tilde{\mathbf{A}} \leftarrow \mathbf{A}$
for $k \in \mathcal{V}$ **do**
  for $i \in \mathcal{V}$ **do**
    for $j \in \mathcal{V}$ **do**
      $\tilde{\mathbf{A}}_{i,j} \mid= \tilde{\mathbf{A}}_{i,k} \;\&\; \tilde{\mathbf{A}}_{k,j}$
Output: $\tilde{\mathbf{A}}$

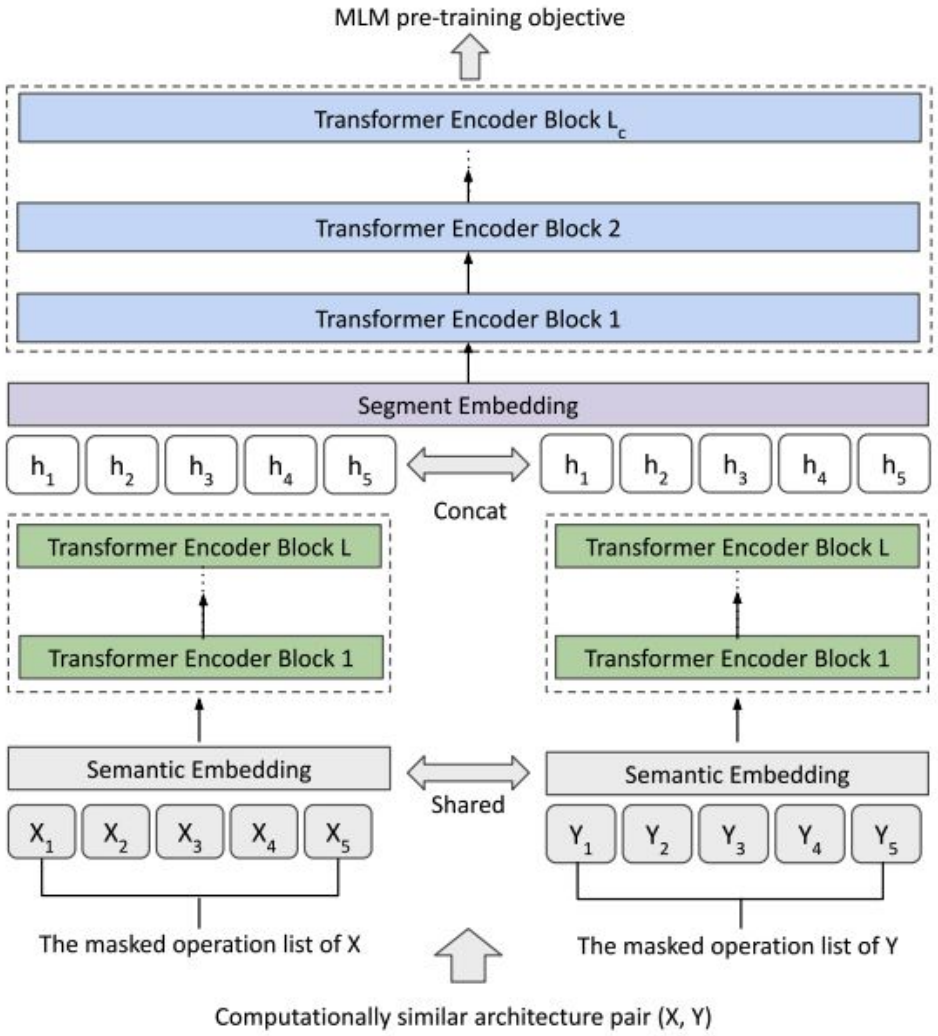
$$\mathbf{M}_{i,j}^{Direct} = \begin{cases} 0, & if \quad A_{i,j} = 1 \\ -\infty, & if \quad A_{i,j} = 0 \end{cases}$$

$$\mathbf{M}_{i,j}^{Indirect} = \begin{cases} 0, & if \quad \tilde{A}_{i,j} = 1 \\ -\infty, & if \quad \tilde{A}_{i,j} = 0 \end{cases}$$

Why:

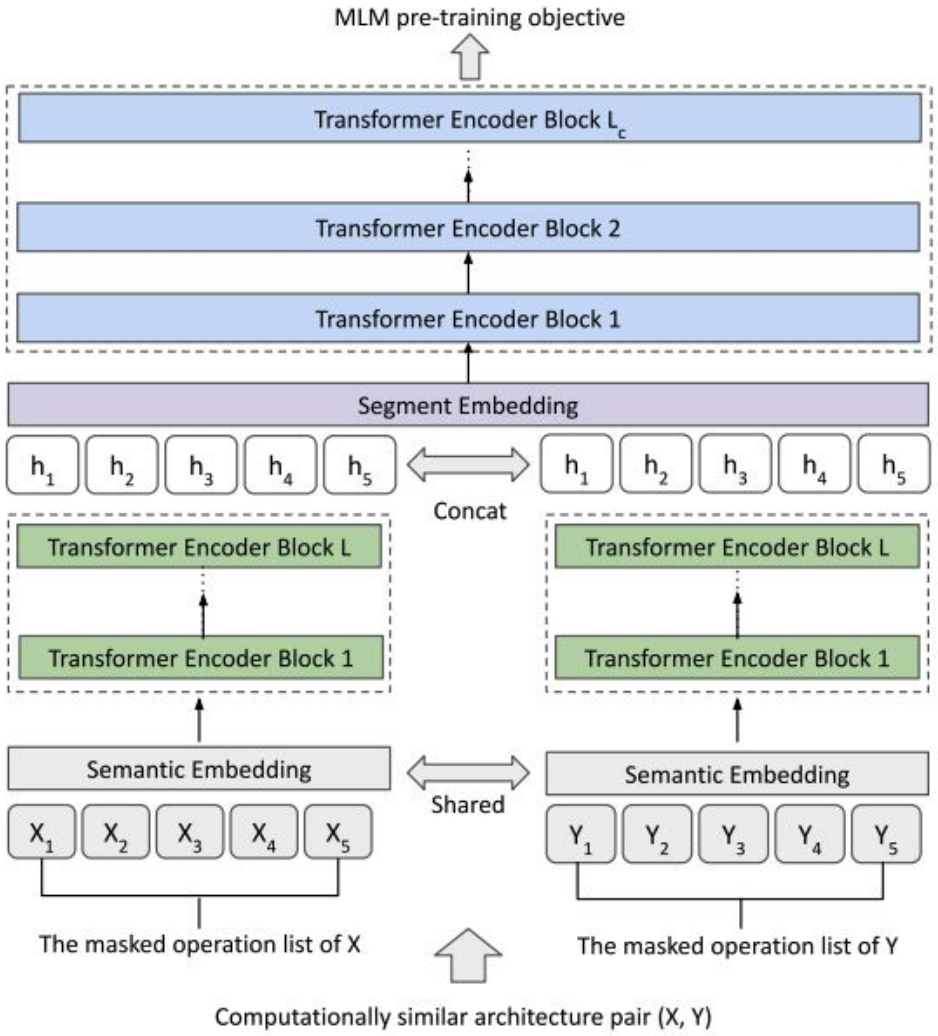
- **Encode long range dependency of different operations beyond adjacency matrix**
- **Together with MLM, encode both local and global computation information**
- **Important for encodings to be generalized to outside search space beyond the training search space**

Pairwise Pre-training Scheme



CATE encodes computationally similar architecture pairs through two Transformers with shared parameters.

Pairwise Pre-training Scheme



The two individual encodings are then concatenated, and the concatenated encoding is fed into another Transformer with a cross-attention encoder to encode the joint information of the architecture pair.

Experimental Setup

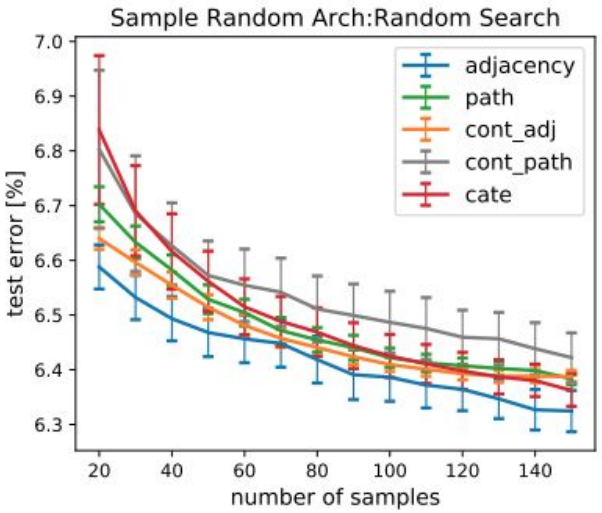
Compare CATE with 11 structure-aware and computation-aware architecture encoding schemes:

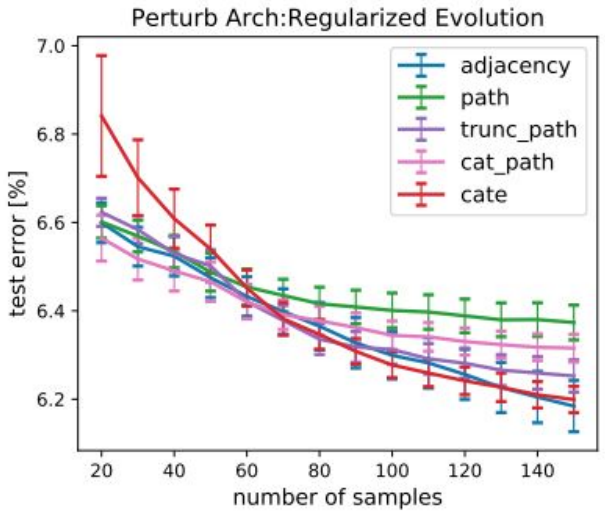
- One-hot/Categorical/Continuous adjacency matrix-based encoding (3)
- One-hot/Categorical/Continuous path-based encoding (3) and their truncated counterparts (3)
- D-VAE
- arch2vec

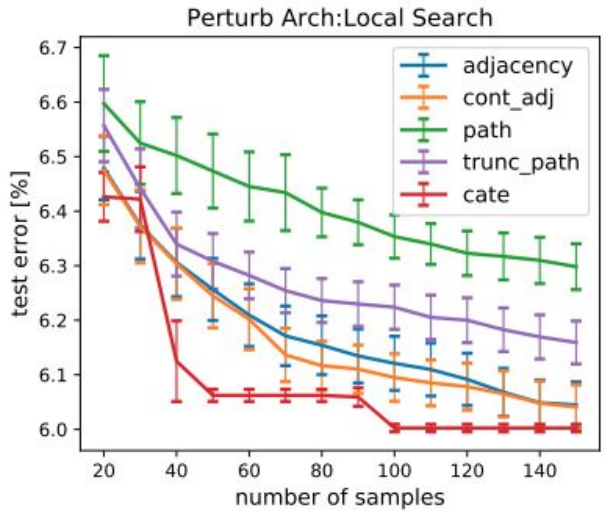
Three major encoding-dependent NAS subroutines:

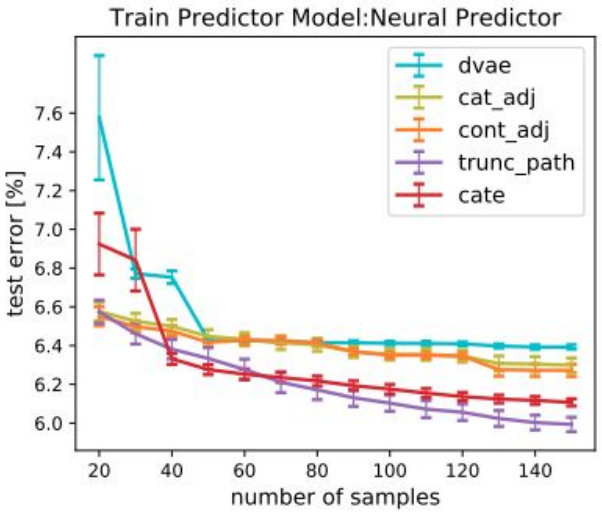
- *Sample random architecture subroutine*: random search (RS)
- *Perturb architecture subroutine*: regularized evolution (REA), local search (LS)
- *Train predictor model subroutine*: neural predictor, BO with GP, BO with DNGO

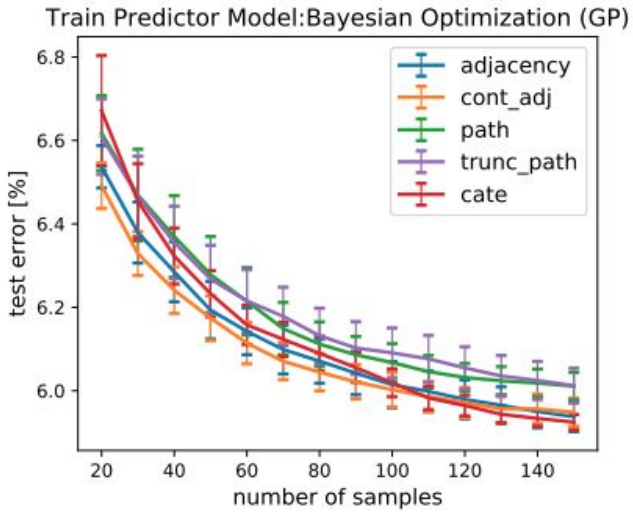
Comparison between CATE and other Encoding Schemes

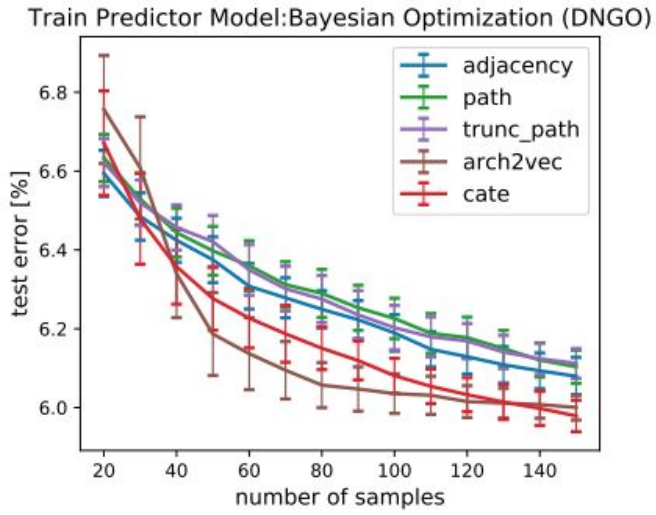




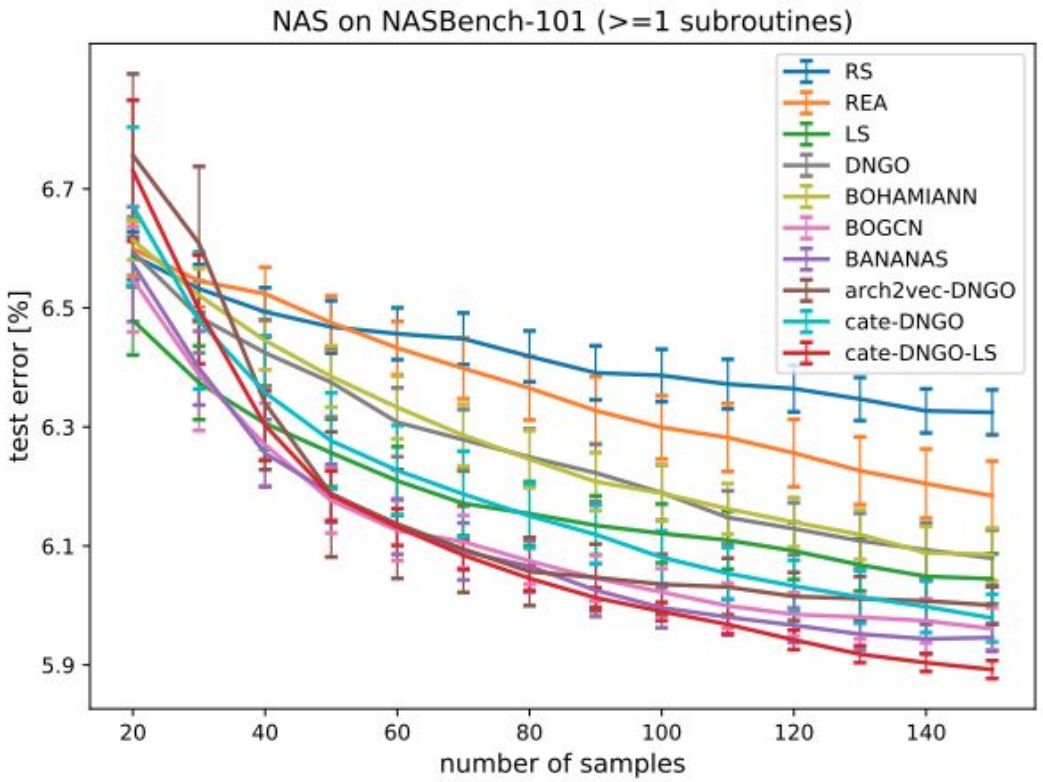


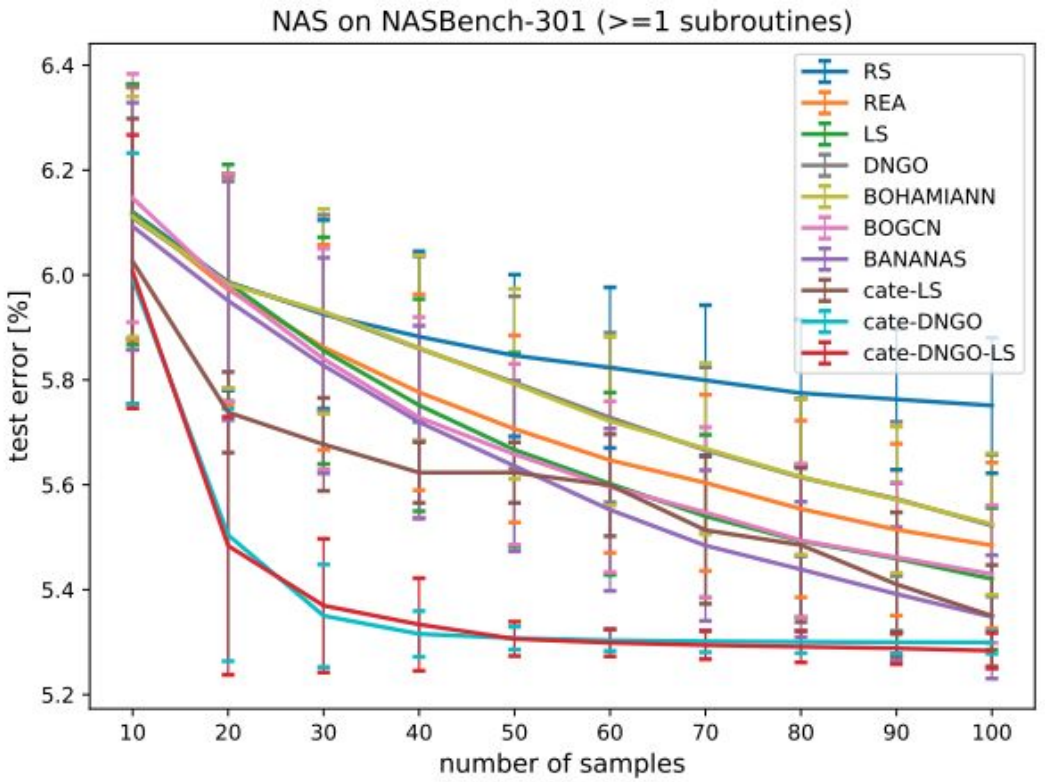






Comparison between CATE and other NAS Methods





Hyperparameter Choices

δ \ K	1	2	4	8
1×10^6	6.02	5.95	5.99	5.95
2×10^6	6.02	**5.94**	6.04	5.96
4×10^6	**5.94**	6.03	6.05	5.99
8×10^6	6.05	6.04	6.11	6.04

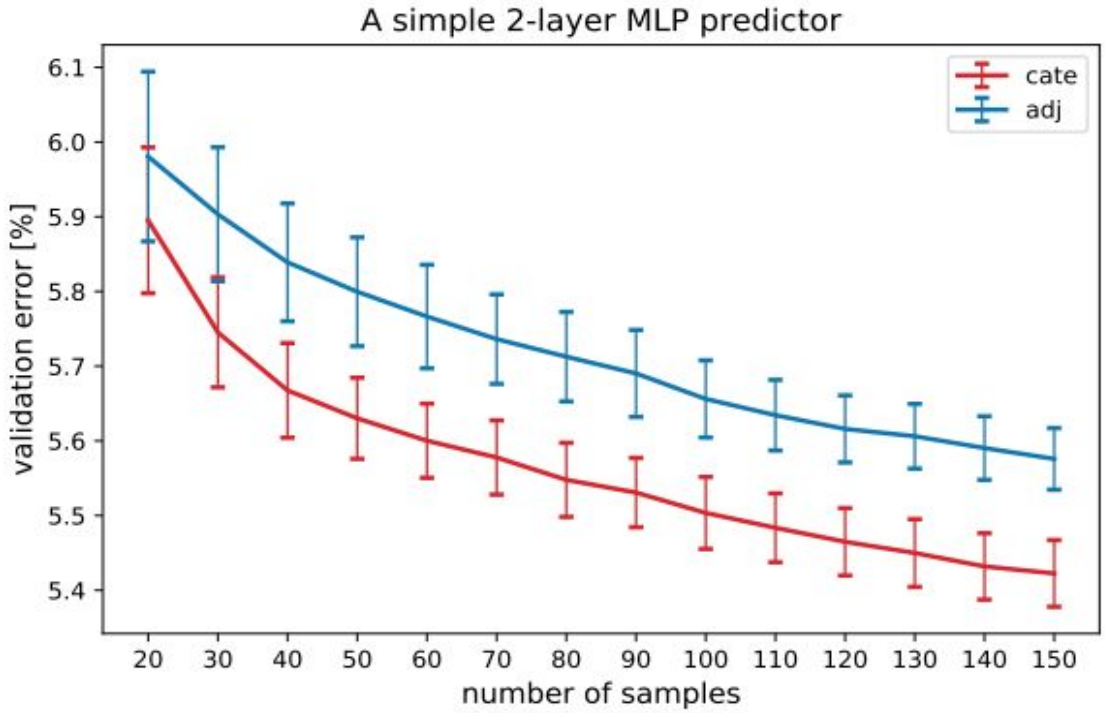
Table 4. Effects of δ and K on architecture pair sampling.

Mask type	NAS-Bench-101	NAS-Bench-301
Direct	6.03	5.35
Indirect	**5.94**	**5.30**

Table 6. Direct/Indirect dependency mask selection.

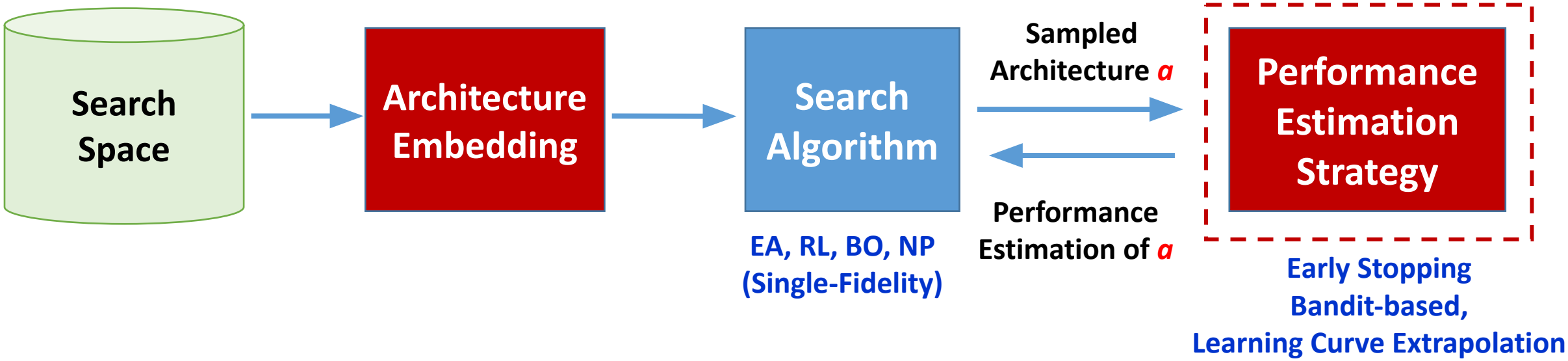
- **Strong computation locality (i.e. small δ) leads to better results**
- **Capturing long-range dependency helps preserve computation information in the encodings**

Generalization to Outside Search Space



- **Better generalization when adapting to outside search space**

Recall this Figure



In the following, I talk about our research on speedy performance estimation.

NAS-Bench-x11 and the Power of Learning Curves [NeurIPS' 21]

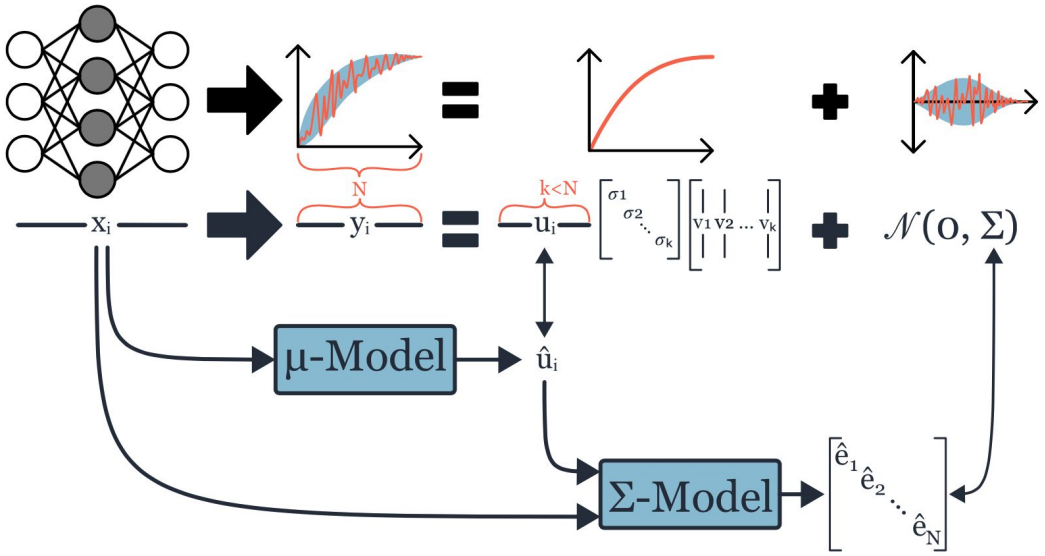
Table 1: Overview of existing NAS benchmarks. We introduce NAS-Bench-111, -311, and -NLP11.

Benchmark	Size	Queryable	Based on	Full train info
NAS-Bench-101	423k	✓		✗
NAS-Bench-201	6k	✓		✓
NAS-Bench-NLP	10^{53}	✗		✗
NAS-Bench-301	10^{18}	✓	DARTS	✗
NAS-Bench-ASR	8k	✓		✓
NAS-Bench-111	423k	✓	NAS-Bench-101	✓
NAS-Bench-311	10^{18}	✓	DARTS	✓
NAS-Bench-NLP11	10^{22}	✓	NAS-Bench-NLP	✓

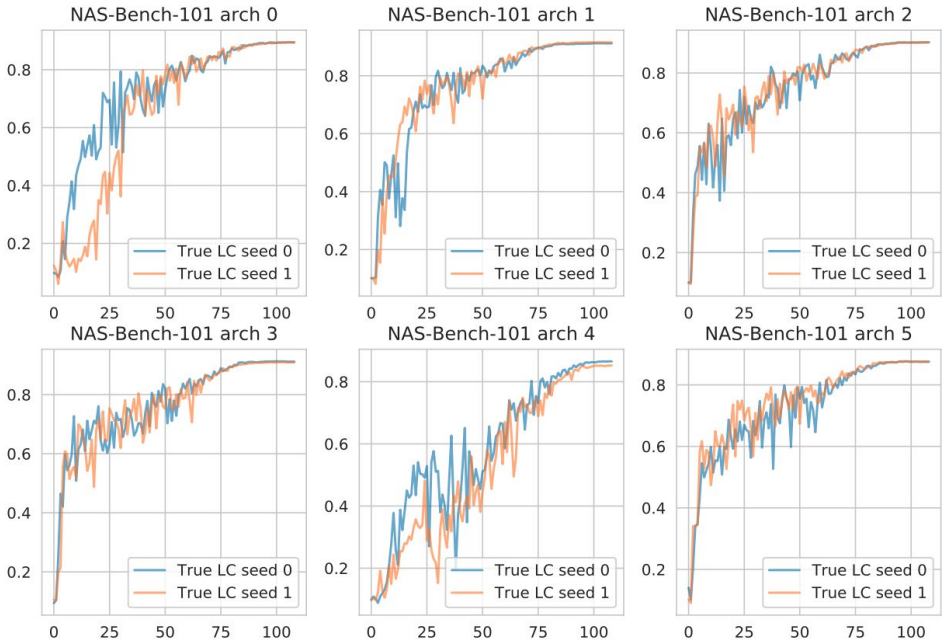
- **A technique to create surrogate NAS benchmarks that include full training information for each architecture, including train/validation/test loss and accuracy learning curves**
- **Allow researchers to easily develop multi-fidelity NAS algorithms**

Roadmap

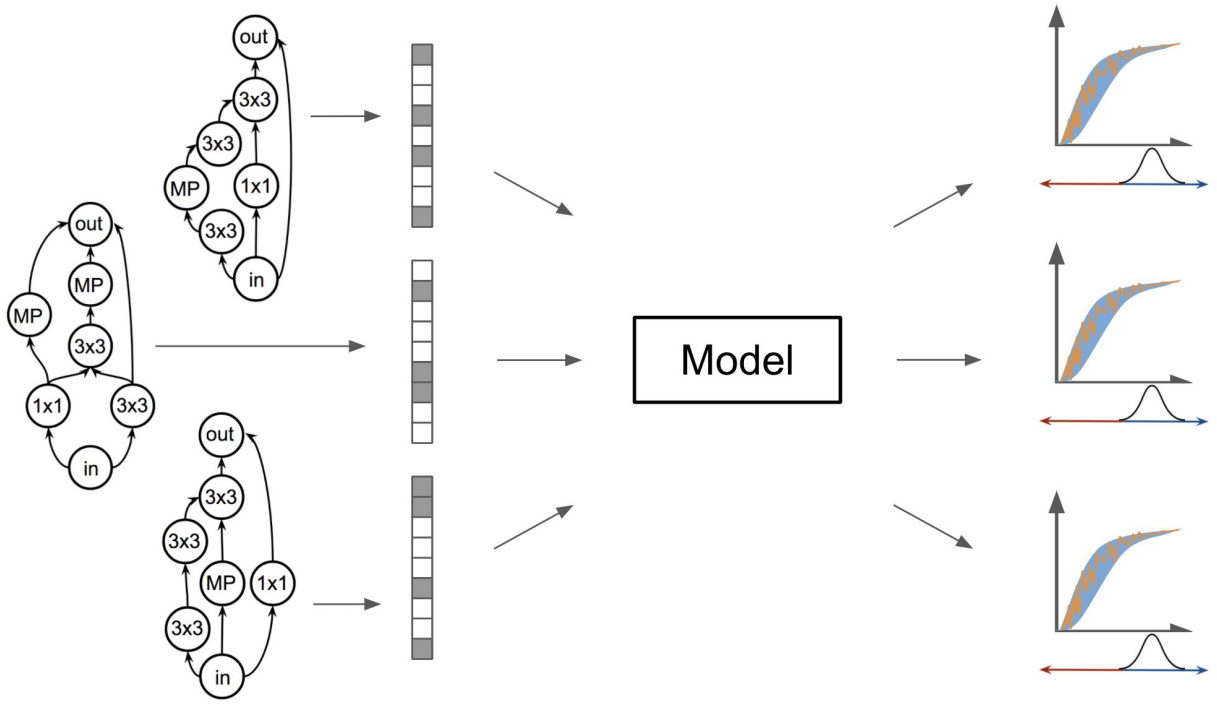
- Generating Learning Curves
- Evaluation
- The Power of Learning Curves



Generating Learning Curves



Generating realistic noise is critical



Goal: given architecture encoding, predict a distribution

Two-part technique

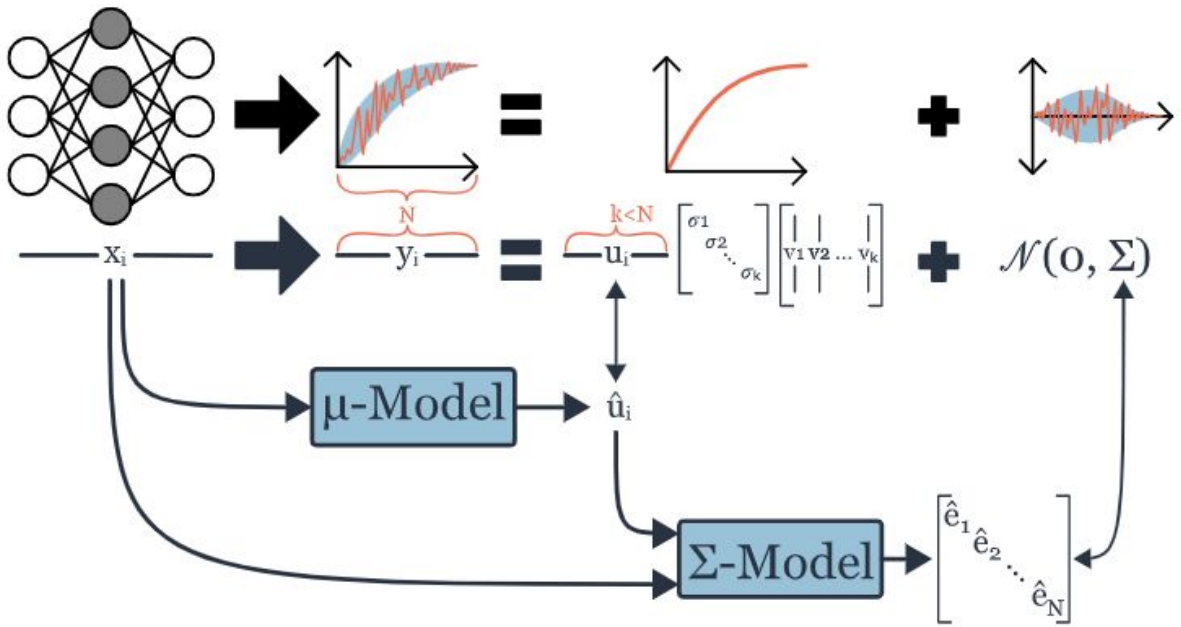
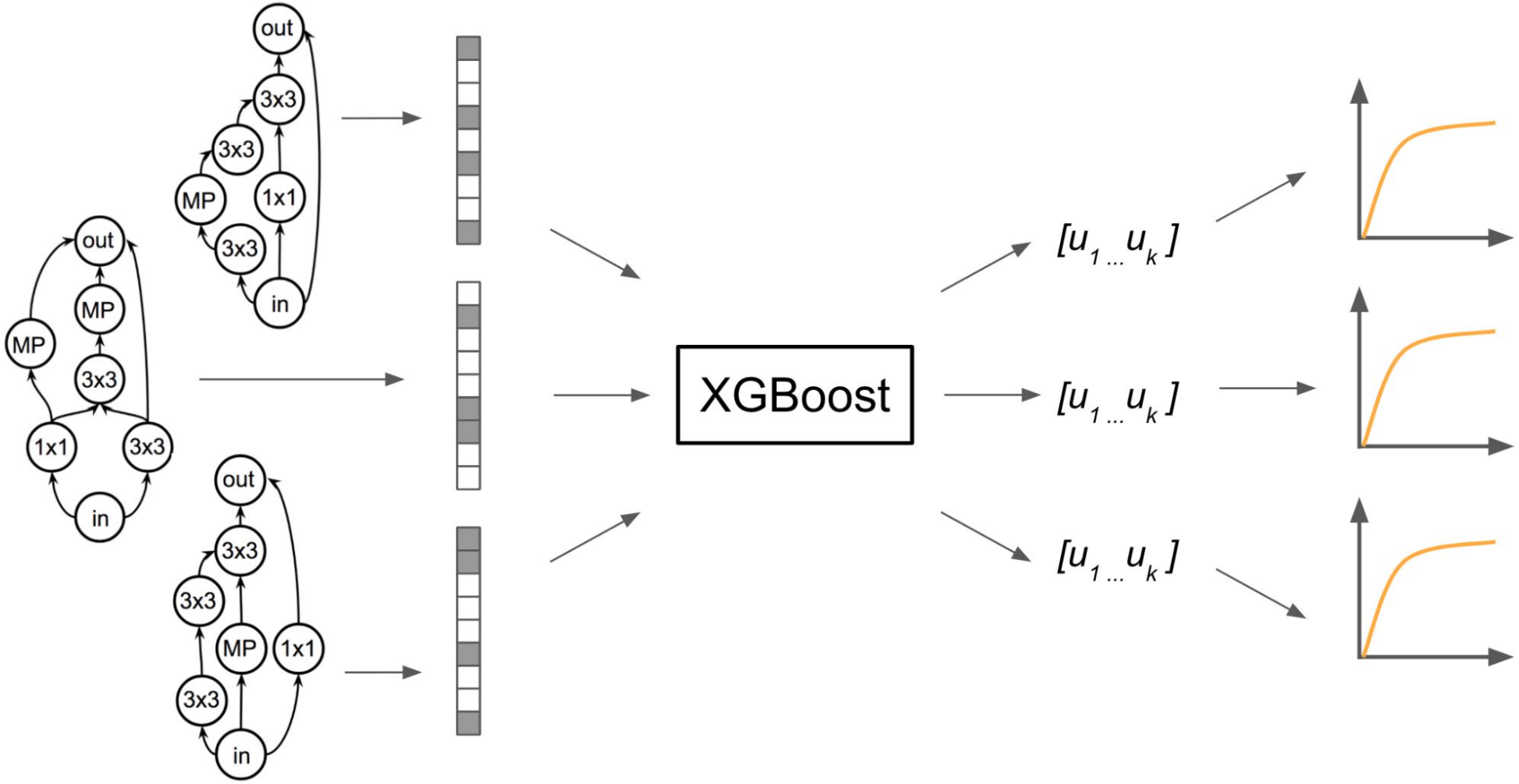


Figure 5: A summary of our approach to create surrogate benchmarks that output realistic learning curves. Compression and decompression functions are learned using the training set of learning curves (in the figure, SVD is shown, but a VAE can also be used). The compression also helps to de-noise the learning curves. A model (μ-model) is trained to predict the compressed (de-noised) learning curves given the architecture encoding. A separate model (Σ-model) is trained to predict each learning curve's noise distribution, given the architecture encoding and predicted compressed learning curve. A realistic learning curve can then be outputted by decompressing the predicted learning curve and sampling noise from the noise distribution.

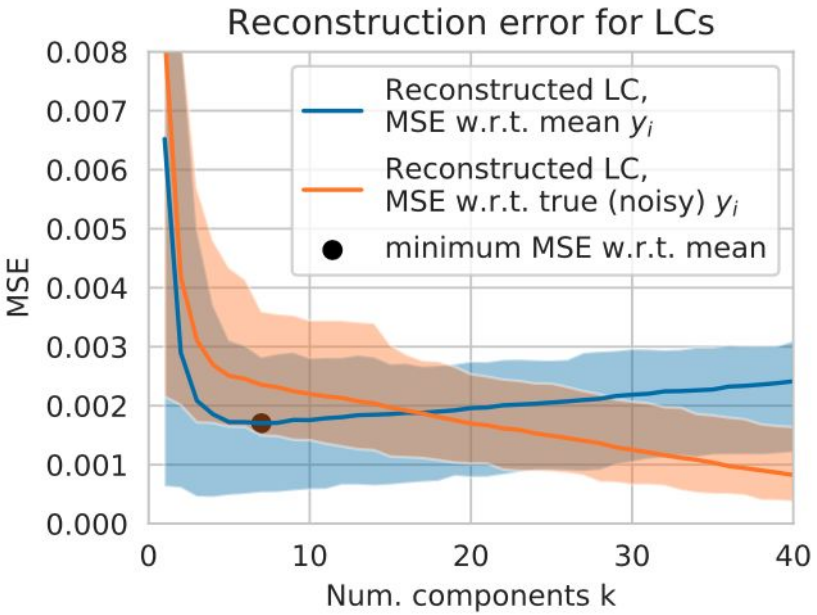
(1) Predict mean LC (2) Predict noise

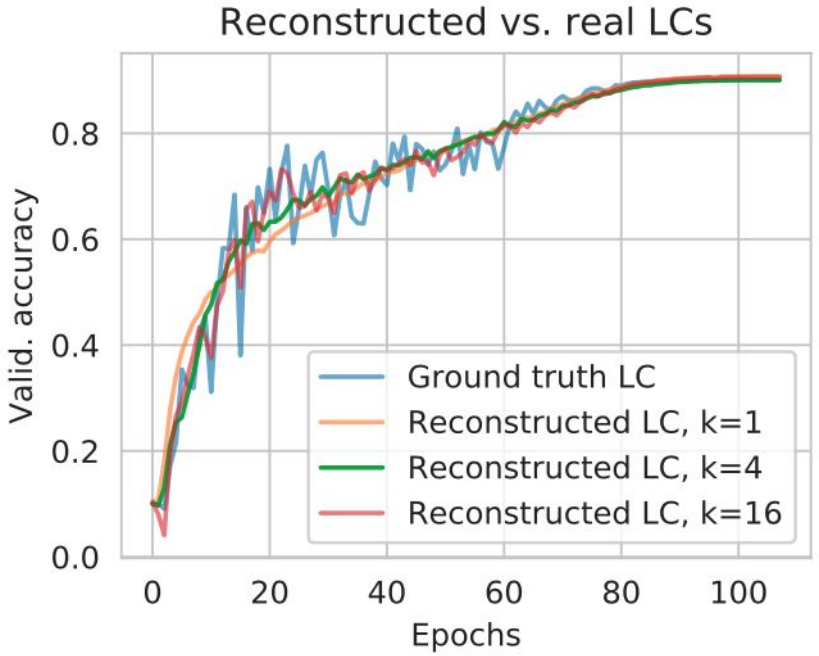
(1) Predicting the mean learning curves



(1) Predicting the mean learning curve

- Compress the learning curves from the training set via SVD
- Predict only the top k principal components





- **SVD helps to reduce the noise**
- **k=6 is the sweet spot where the compression function minimizes reconstruction error without overfitting the noise of individual learning curves**

(2) Noise Modeling

- Assume the noise comes from an isotropic Gaussian distribution
  - The noise distribution is the same for all architectures
    - a simple sample standard deviation statistic **(STD)**
    - a Gaussian kernel density estimation **(GKDE)** model trained on residuals to create a multivariate KDE
  - For each architecture, the noise in a small window of epochs are i.i.d.
    - a model trained on estimate the distribution of noise over **a window of epochs**

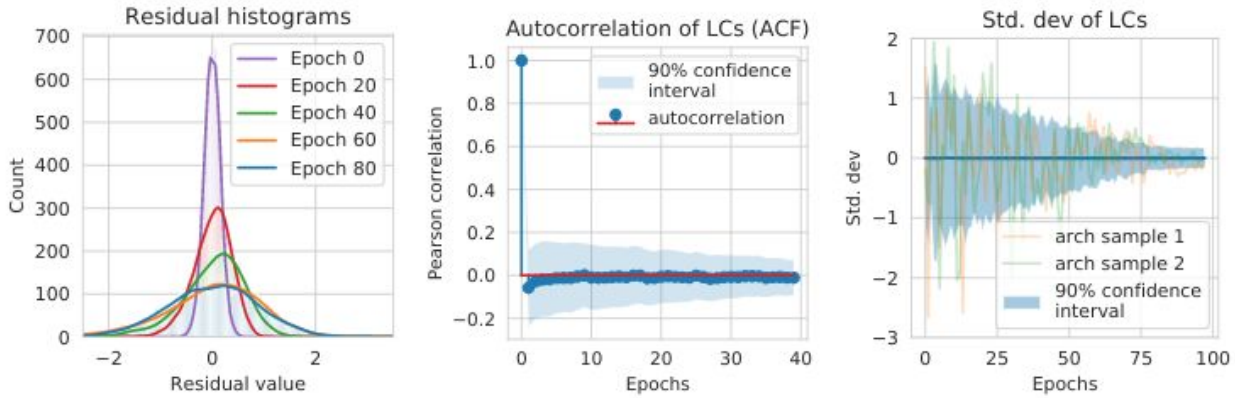
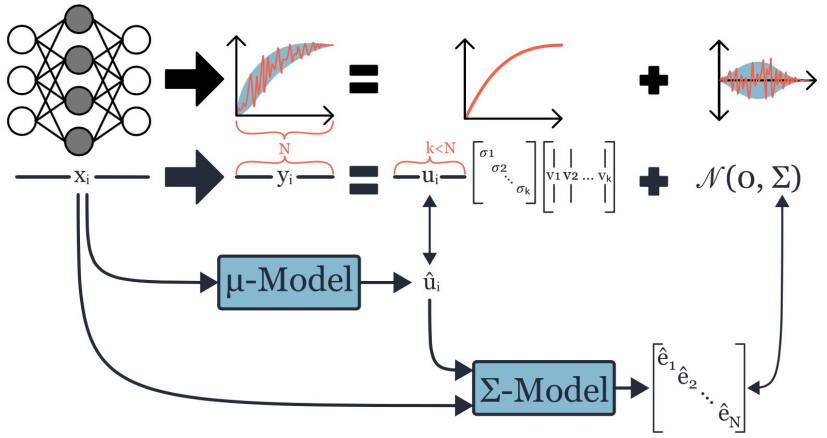


Figure 6: A plot of the residuals across all architectures for five different epochs (left). We see that the distributions are roughly Gaussian. A plot of the autocorrelation function (ACF) averaged over all training learning curves (middle). We see that there is only a small amount of autocorrelation. A plot of the 90% confidence intervals of the residuals at each epoch (right). All plots use the NAS-Bench-301 learning curve training set.



Surrogate Selection

- Full ablation study on 18 combinations {SVD, VAE}, {LGB, XGB, MLP}, {GKDE, STD, Sliding Window}
- SVD-LGB-GKDE archives the best performance
- Therefore, we use it as the surrogate predictor to extrapolate the learning curves

Coefficient of determination: R^2

Kendall Tau rank correlation: KT

Kullback Leibler divergence: KL

Table 3: Evaluation of the surrogate benchmarks on test sets, with all combinations of models. For NAS-Bench-111 and NAS-Bench-NLP11, we use architecture accuracies as additional features to improve performance. As explained in Section B, no architectures in the NAS-Bench-NLP dataset were trained more than once, so we do not compute KL divergence for NAS-Bench-NLP11.

Benchmark	Avg. R^2	Final R^2	Avg. KT	Final KT	Avg. KL	Final KL
NAS-Bench-111						
SVD-LGB-GKDE	**0.630**	**0.853**	**0.611**	**0.794**	**1.641**	0.516
SVD-LGB-STD	**0.630**	**0.853**	**0.611**	**0.794**	2.768	**0.383**
SVD-LGB-window	**0.630**	**0.853**	**0.611**	**0.794**	24.402	3.303
SVD-XGB-GKDE	0.329	0.378	0.408	0.429	2.743	0.580
SVD-XGB-STD	0.329	0.378	0.408	0.429	4.867	0.503
SVD-XGB-window	0.329	0.378	0.408	0.429	38.457	16.172
SVD-MLP-GKDE	0.195	0.065	0.330	0.290	4.599	0.762
SVD-MLP-STD	0.195	0.065	0.330	0.290	8.417	0.848
SVD-MLP-window	0.195	0.065	0.330	0.290	82.180	15.711
VAE-LGB-GKDE	0.267	0.218	0.462	0.617	3.788	0.829
VAE-LGB-STD	0.267	0.218	0.462	0.617	6.866	0.972
VAE-LGB-window	0.267	0.218	0.462	0.617	53.866	19.820
VAE-XGB-GKDE	0.311	0.272	0.453	0.559	3.828	0.828
VAE-XGB-STD	0.311	0.272	0.453	0.559	6.940	0.969
VAE-XGB-window	0.311	0.272	0.453	0.559	55.654	19.614
VAE-MLP-GKDE	0.218	0.007	0.386	0.369	4.583	0.844
VAE-MLP-STD	0.218	0.007	0.386	0.369	8.386	1.001
VAE-MLP-window	0.218	0.007	0.386	0.369	83.481	19.091
NAS-Bench-311						
SVD-LGB-GKDE	**0.779**	**0.800**	**0.728**	**0.788**	**0.503**	**0.548**
SVD-LGB-STD	**0.779**	**0.800**	**0.728**	**0.788**	0.919	1.036
SVD-LGB-window	**0.779**	**0.800**	**0.728**	**0.788**	1.566	4.083
SVD-XGB-GKDE	0.522	0.546	0.607	0.654	1.783	3.272
SVD-XGB-STD	0.522	0.546	0.607	0.654	3.271	5.958
SVD-XGB-window	0.522	0.546	0.607	0.654	5.282	19.432
SVD-MLP-GKDE	0.564	0.549	0.573	0.603	15.727	29.057
SVD-MLP-STD	0.564	0.549	0.573	0.603	28.833	52.515
SVD-MLP-window	0.564	0.549	0.573	0.603	45.071	167.140
VAE-LGB-GKDE	0.431	0.447	0.568	0.616	5.995	13.486
VAE-LGB-STD	0.431	0.447	0.568	0.616	11.015	24.836
VAE-LGB-window	0.431	0.447	0.568	0.616	17.510	79.773
VAE-XGB-GKDE	0.397	0.427	0.577	0.624	6.520	16.739
VAE-XGB-STD	0.397	0.427	0.577	0.624	11.978	30.368
VAE-XGB-window	0.397	0.427	0.577	0.624	18.883	97.485
VAE-MLP-GKDE	0.509	0.520	0.584	0.619	13.545	33.851
VAE-MLP-STD	0.509	0.520	0.584	0.619	24.770	61.455
VAE-MLP-window	0.509	0.520	0.584	0.619	38.593	196.246
NAS-Bench-NLP11						
SVD-LGB	0.878	**0.895**	**0.878**	**0.844**	-	-
SVD-XGB	0.877	0.806	0.856	0.820	-	-
SVD-MLP	**0.893**	0.856	0.742	0.692	-	-
VAE-LGB	0.862	0.847	0.766	0.770	-	-
VAE-XGB	0.875	0.860	0.720	0.687	-	-
VAE-MLP	0.867	0.871	0.667	0.685	-	-

The Power of Learning Curve Extrapolation

Algorithm 1 Single-Fidelity Algorithm

```
initialize history
while t < t_max :
    arches = gen_candidates(history)
    accs = train(arches, epoch=E_max)
    history.update(arches, accs)
Return arch with the highest acc
```

Algorithm 2 LCE Framework

```
initialize history
while t < t_max :
    arches = gen_candidates(history)
    accs = train(arches, epoch=E_few)
    sorted_by_pred = LCE(arches, accs)
    arches = sorted_by_pred[:top_n]
    accs = train(arches, epoch=E_max)
    history.update(arches, accs)
Return arch with the highest acc
```

- **Propose a simple learning curve extrapolation (LCE) framework to speed-up NAS**
- **Extrapolator candidates:**
  - **Model-free:**
    - Weighted Probabilistic Modeling (WPM): using MCMC to sample the most promising fit
  - **Model-based:**
    - Learning Curve Support vector regressor (LcSVR)

LCE Applied to a Single-Fidelity

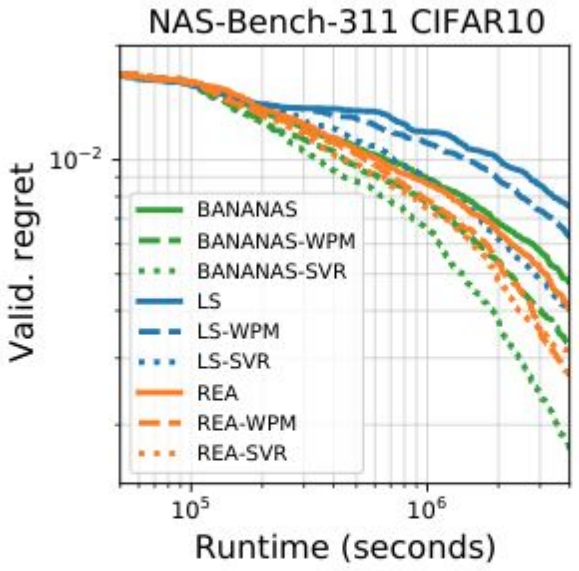


Figure 3: LCE Framework applied to single-fidelity algorithms on NAS-Bench-311.

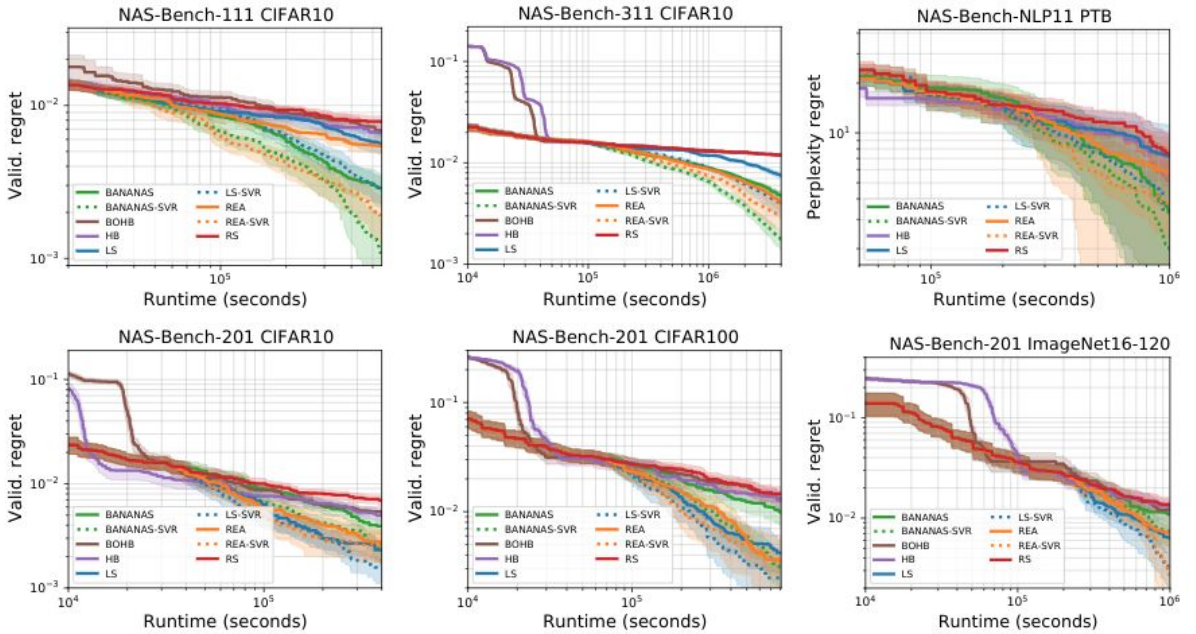
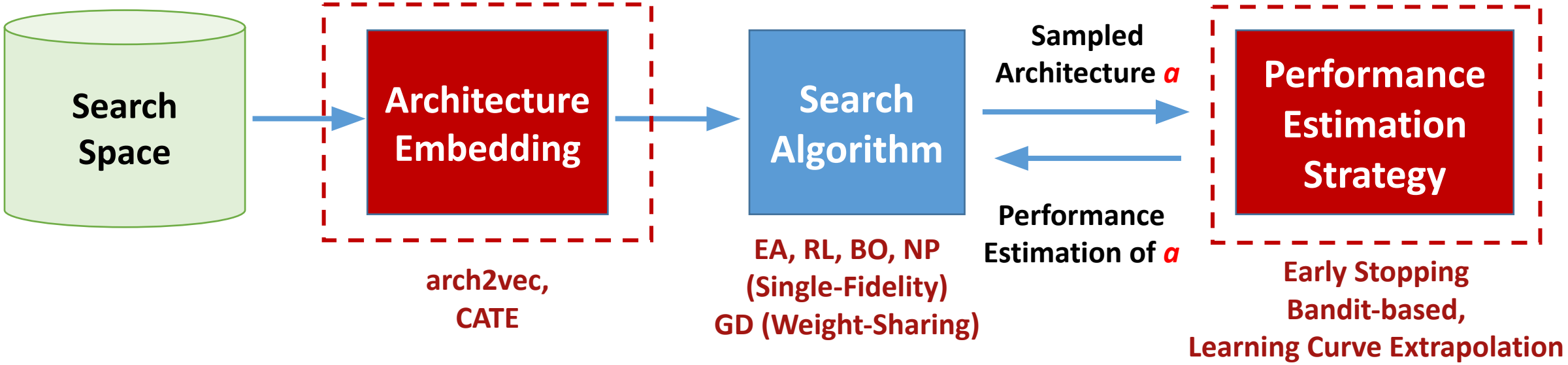


Figure 4: NAS results on six different combinations of search spaces and datasets. For every setting, an SVR augmented method performs best.

- Solid line: Best sampling-based NAS methods (e.g. BANANAS, LS, REA)
- Dashed line: Further improvement via LcSVR or WPM
- **Consistent improvement across all search spaces**

Summary of my research on NAS



Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, Neural Architecture Search: A Survey, JMLR 2019

Papers and Codes

For more detailed information and codes, please refer to:

arch2vec [NeurIPS' 20]: https://arxiv.org/abs/2006.06936
https://github.com/MSU-MLSys-Lab/arch2vec

CATE [ICML' 21]: https://arxiv.org/abs/2102.07108
https://github.com/MSU-MLSys-Lab/CATE

NAS-Bench-x11 [NeurIPS' 21]: https://arxiv.org/abs/2111.03602
https://github.com/automl/nas-bench-x11
https://github.com/automl/NASLib