AutoML for Deep Recommender Systems: Fundamentals and Advances

Ruiming Tang
Huawei Noah's Ark Lab
tangruiming@huawei.com

Bo Chen
Huawei Noah's Ark Lab
chenbo116@huawei.com

Yejing Wang
City University of Hong Kong
adave631@gmail.com

Huifeng Guo
Huawei Noah's Ark Lab
huifeng.guo@huawei.com

Yong Liu
Huawei Noah's Ark Lab
liu.yong6@huawei.com

Wenqi Fan
The Hong Kong Polytechnic University
wenqifan03@gmail.com

Xiangyu Zhao
City University of Hong Kong
xianzhao@cityu.edu.hk

Tutorial Website (Slides): https://advanced-recommender-systems.github.io/AutoML-Recommendations/

A Comprehensive Survey on Automated Machine Learning for Recommendations, arXiv:2204.01390

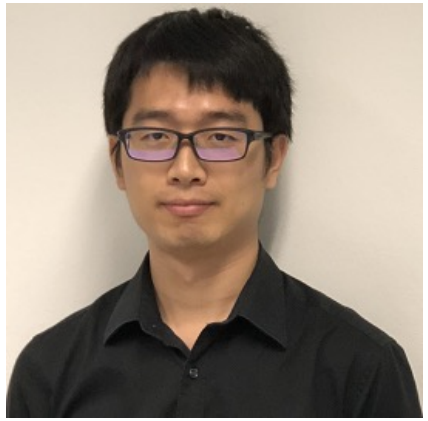
Presenter Bio

Ruiming Tang
Lab director
Huawei Noah’s Ark Lab

Bo Chen
Researcher
Huawei Noah’s Ark Lab

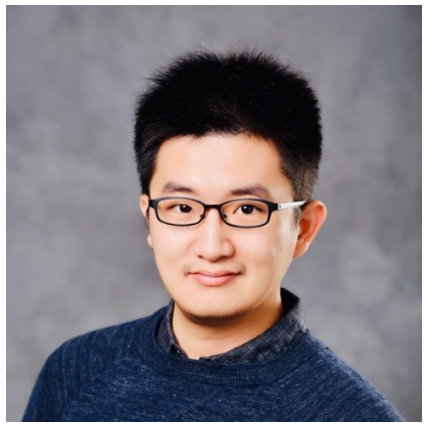
Yejing Wang
Ph.D. student
City University of Hong Kong

Huifeng Guo
Researcher
Huawei Noah’s Ark Lab



Yong Liu
Researcher
Huawei Noah’s Ark Lab

Wenqi Fan
Professor
Hong Kong Polytechnic University

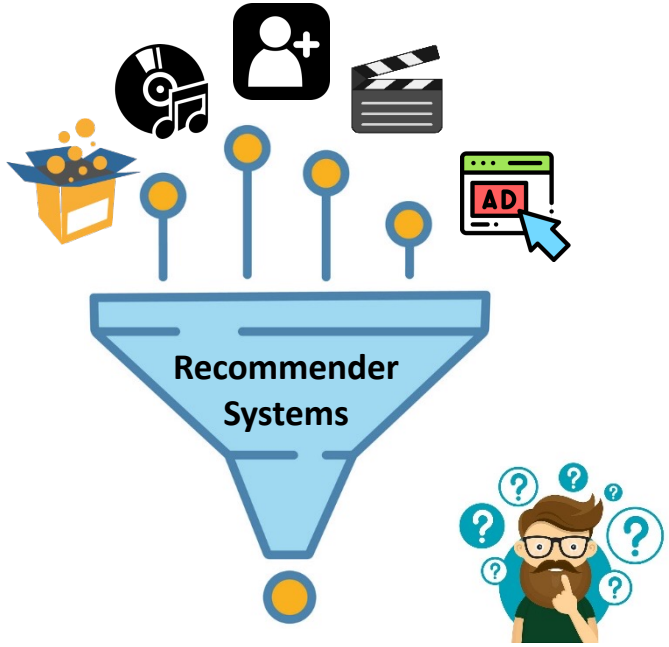


Xiangyu Zhao
Professor
City University of Hong Kong

Recommender Systems

Age of Information Explosion

Information overload



Recommend item X to user

Items can be Products, News, Movies, Videos, Friends, etc.

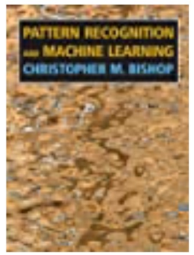
Recommender Systems

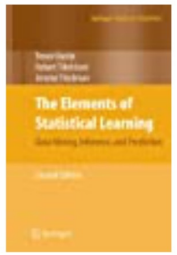
- Recommendation has been widely applied in online services
  - **E-commerce**, Content Sharing, Social Networking, etc.

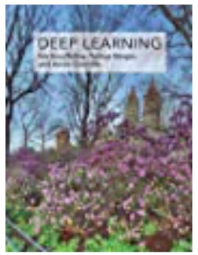
amazon ebay 淘宝网 Taobao.com JD.COM

Product Recommendation

Frequently bought together



A + 

B + 

C

Total price: $208.9

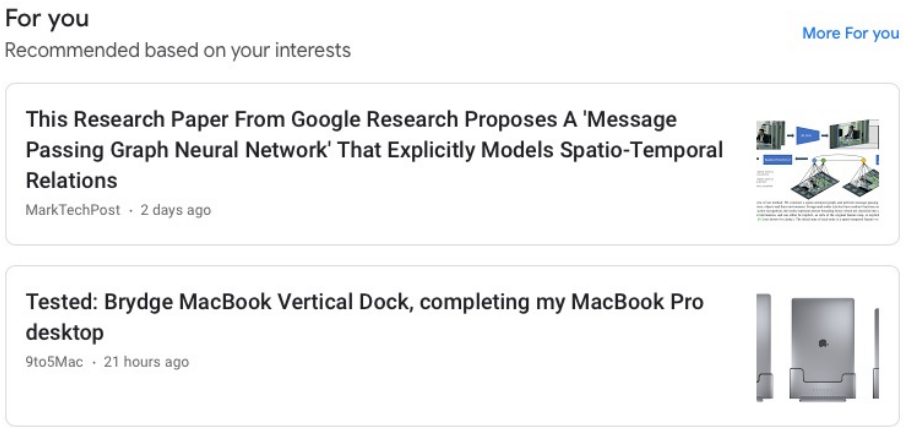
Add all three to Cart

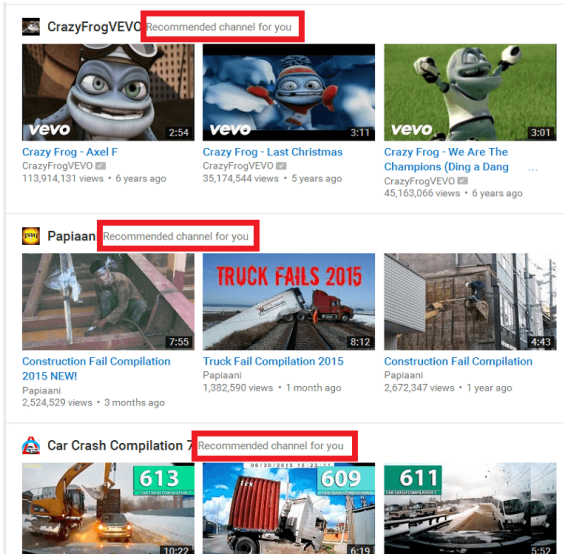
Add all three to List

Recommender Systems

- Recommendation has been widely applied in online services
  - E-commerce, **Content Sharing**, Social Networking, etc.

News/Video/Image Recommendation





Recommender Systems

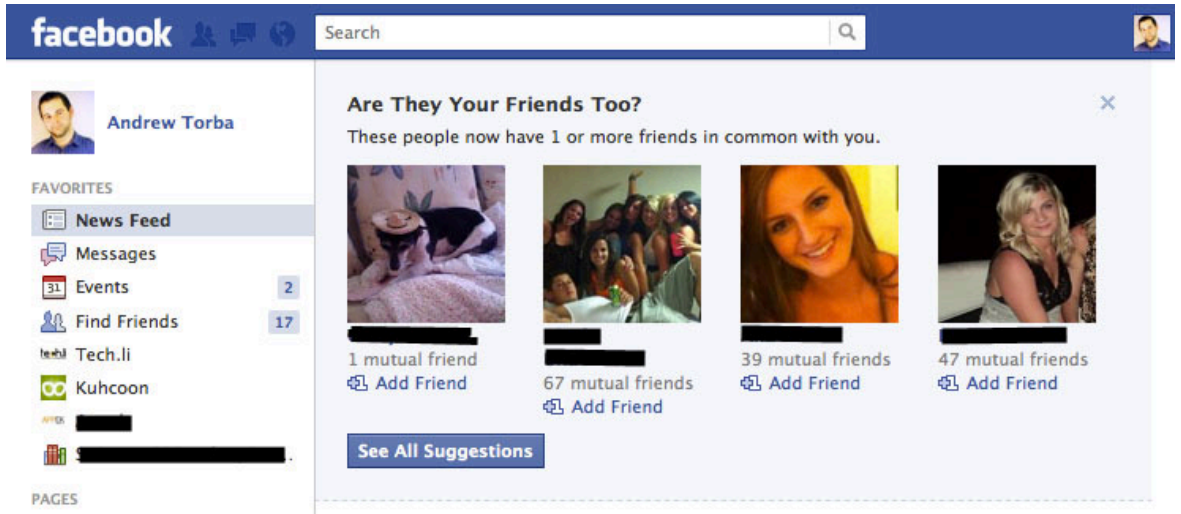
- Recommendation has been widely applied in online services
  - E-commerce, Content Sharing, **Social Networking**, etc.

facebook

新浪微博 weibo.com

LinkedIn

Friend Recommendation



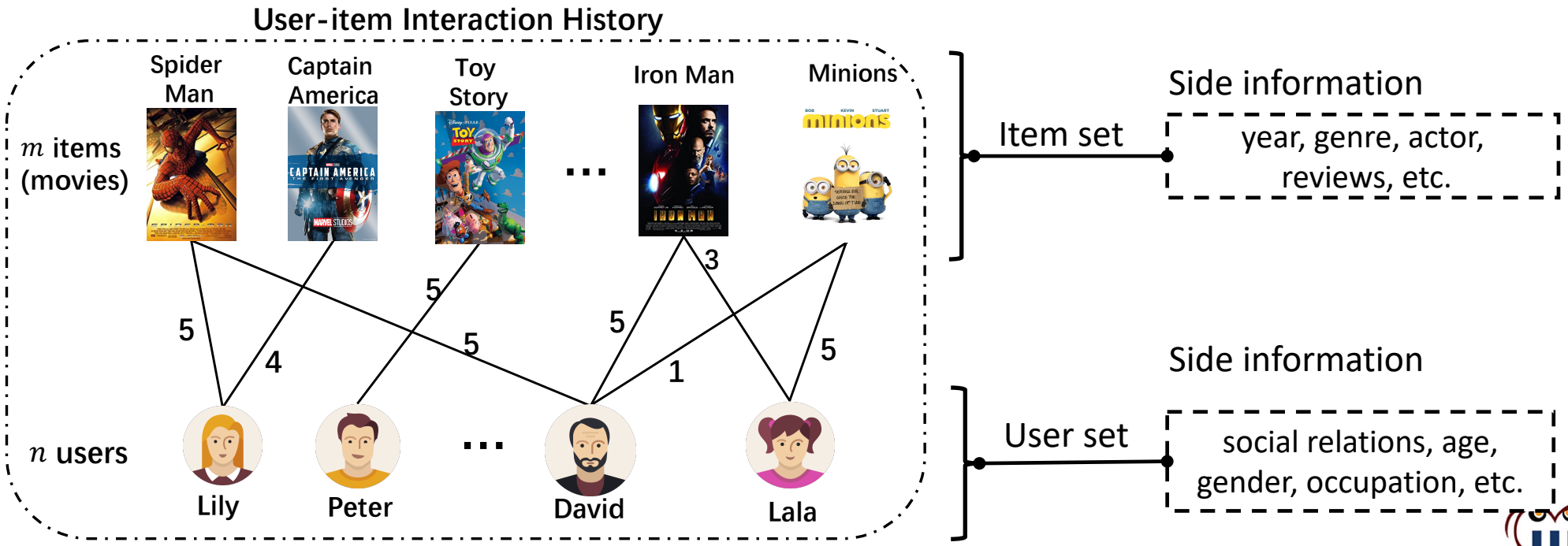
Problem Formulation

INPUT

Historical user-item interactions or additional side information (e.g., social relations, item's knowledge, etc.)

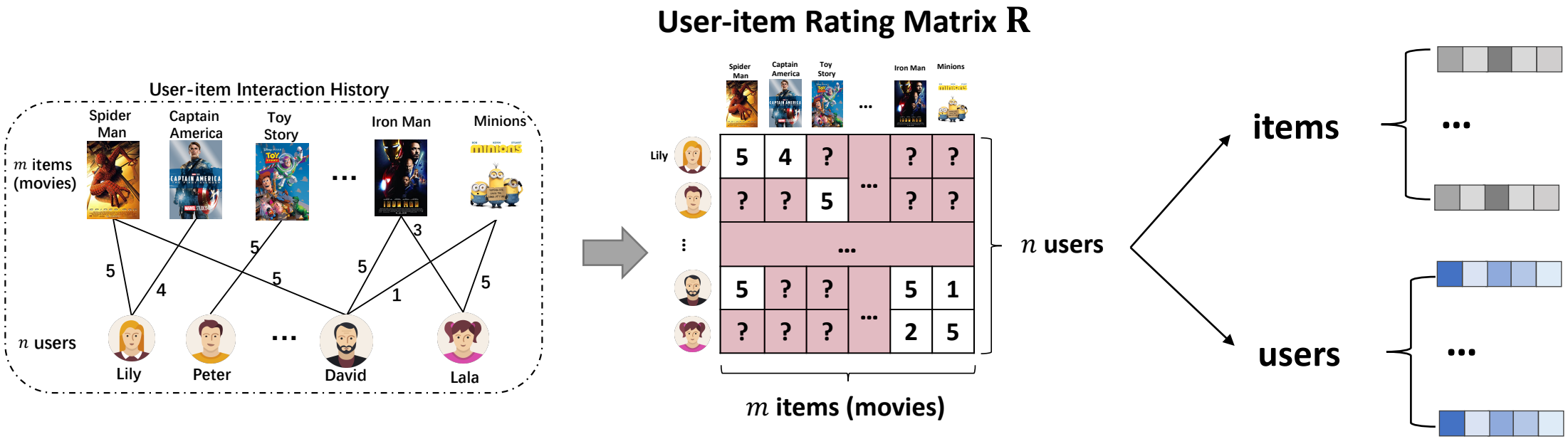
OUTPUT

Predict how likely a user would interact with a target Item (e.g., click, view, or purchase)



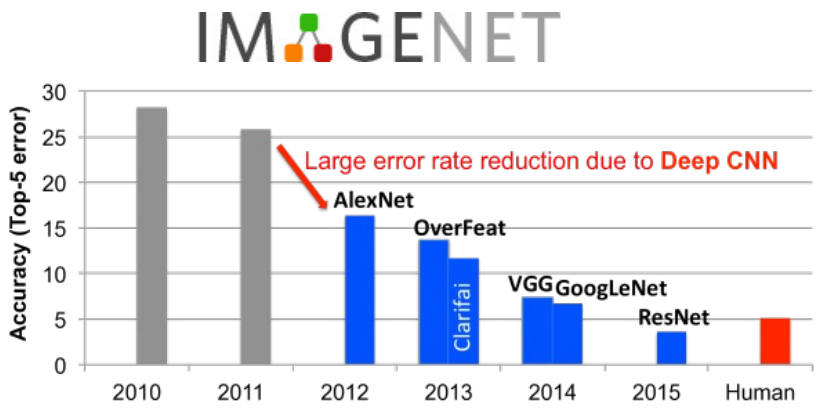
Recommender Systems

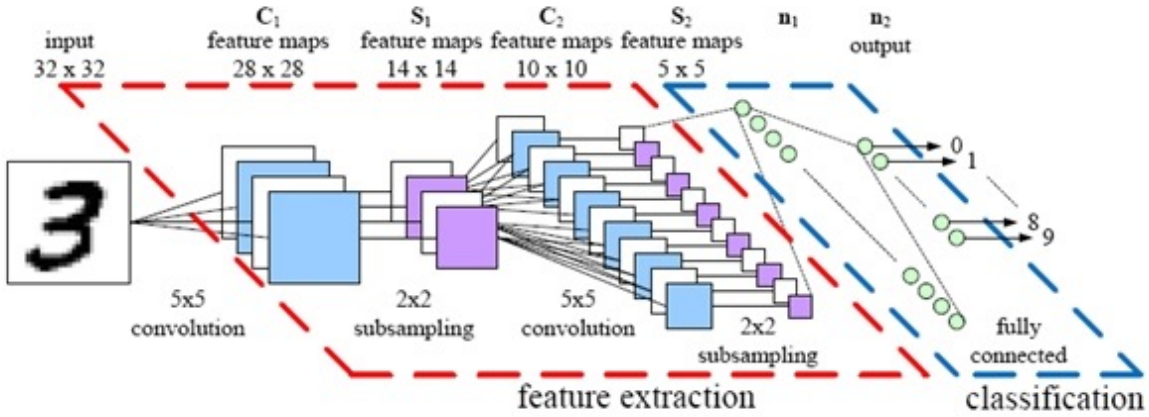
- Collaborative Filtering (CF) is the most well-known technique for recommendation
  - Similar users (with respect to their historical interactions) have similar preferences
  - Modelling users' preference on items based on their past interactions (e.g., ratings and clicks)
- Learning representations of users and items is the key of CF

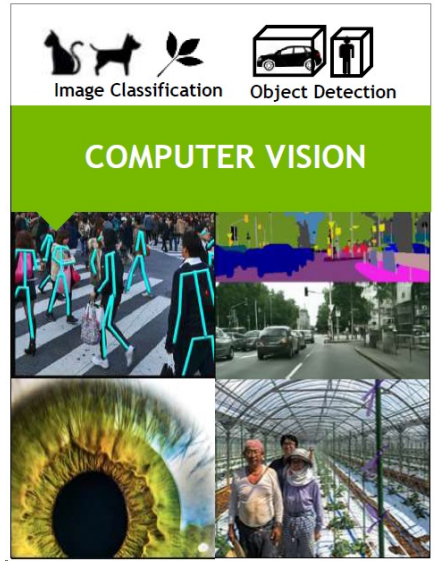


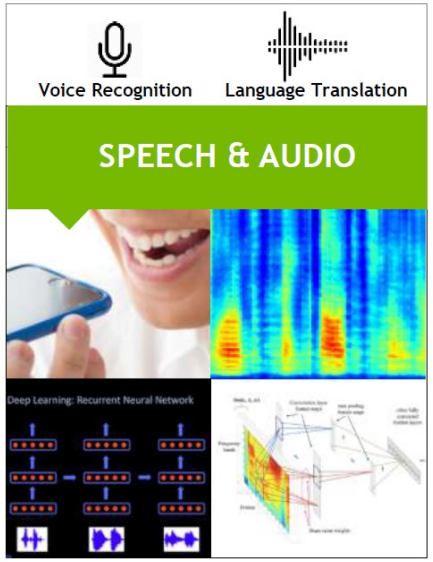
Task: predicting missing movie ratings in Netflix.

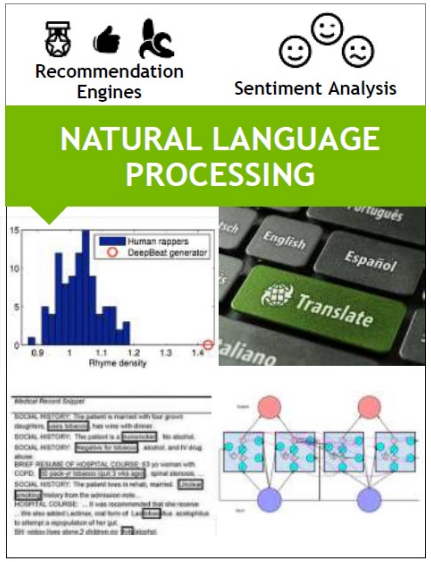
Deep Learning is Changing Our Lives





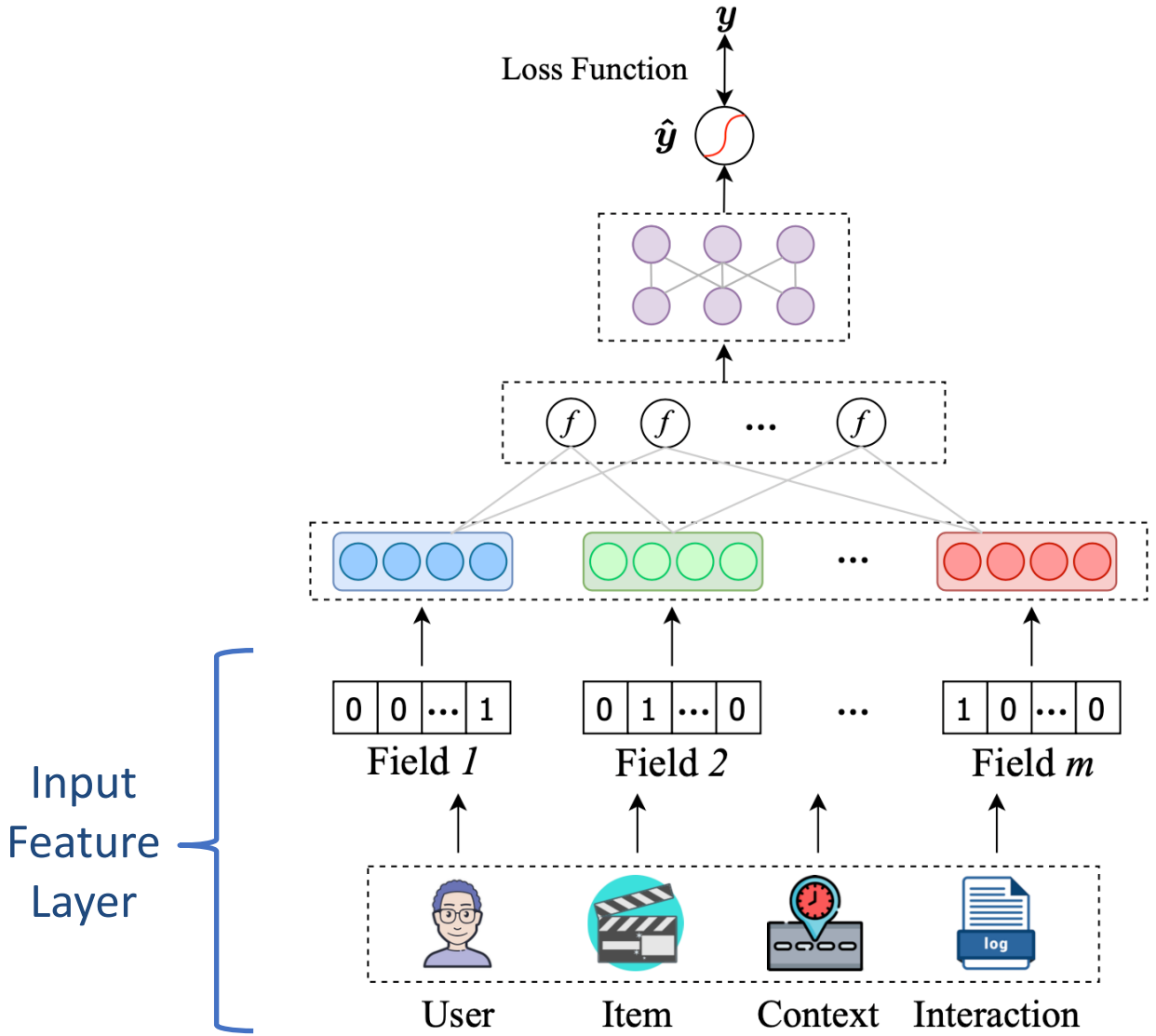






wsdm 2023
SINGAPORE FEBRUARY 27 - MARCH 3

Deep Recommender Architecture

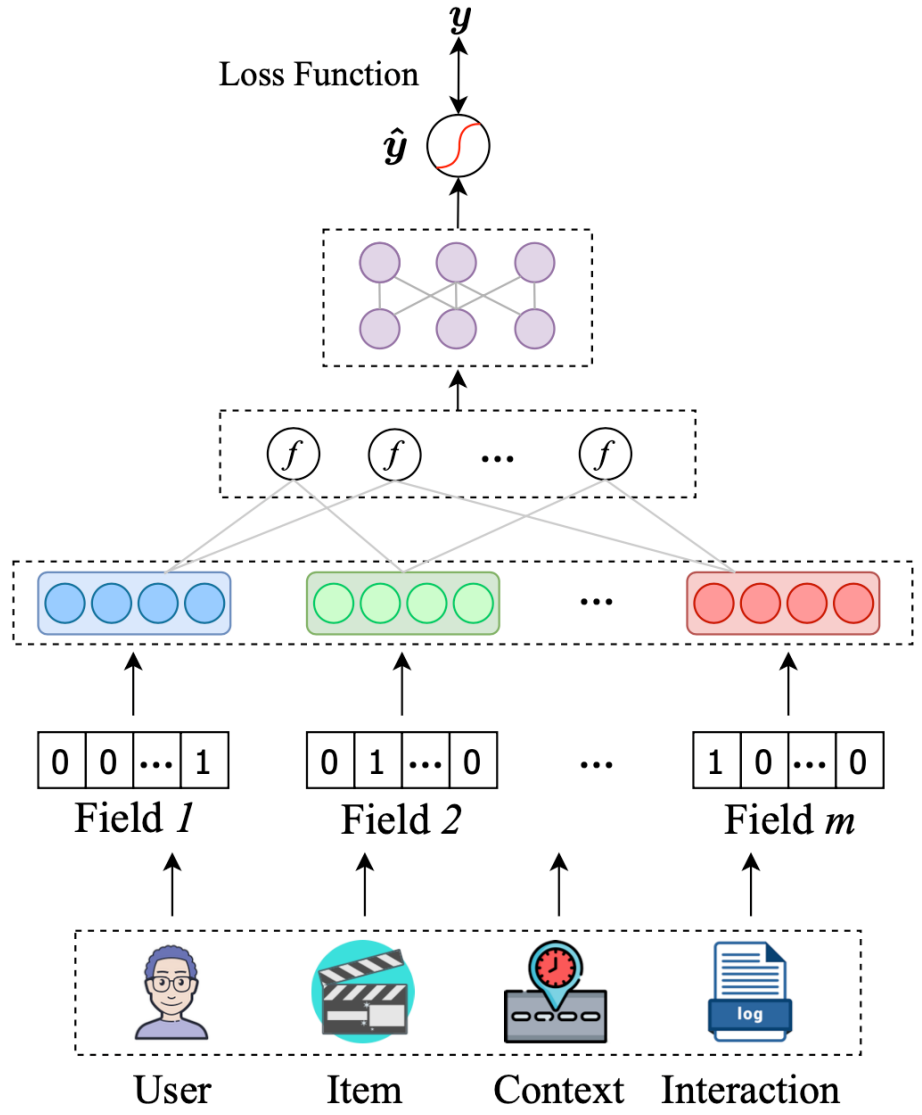


Deep Recommender Architecture

Output Layer

Feature Interaction Layer

Feature Embedding Layer



System Design

Deep Recommender Architecture

Output Layer

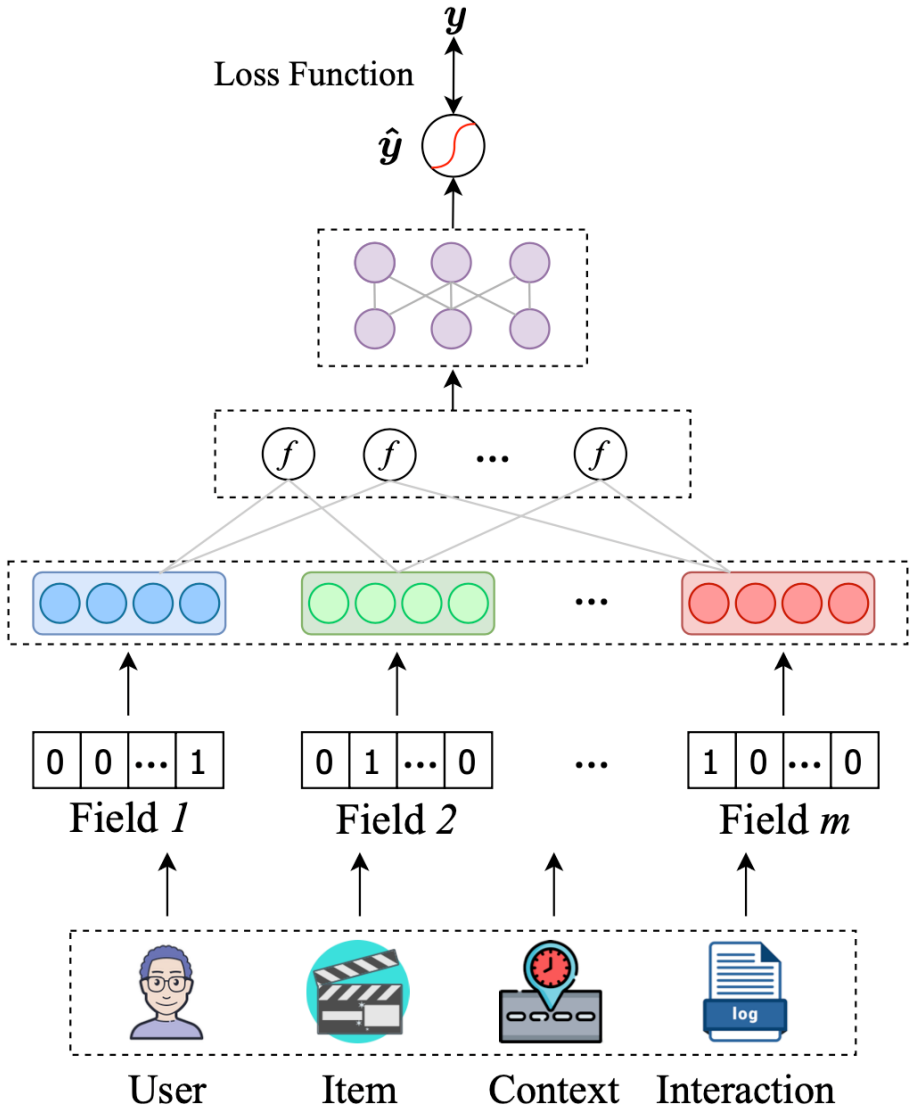
BCE, BPR, MSE

Feature Interaction Layer

Pooling, convolution, and the number of layers, inner product, outer product, convolution, etc.

Feature Embedding Layer

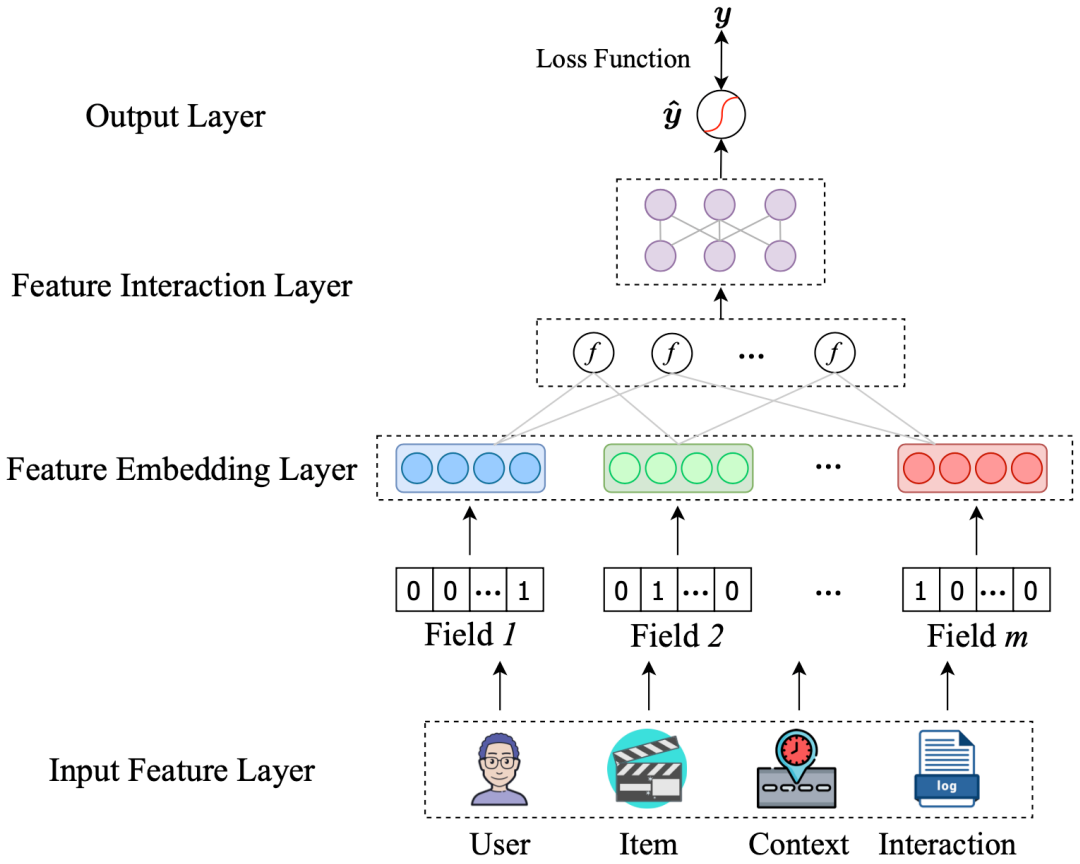
High/low-frequency features embedding sizes



System Design

Hardware infrastructure, data pipeline, information transfer, implementation, deployment, optimization, evaluation, etc.

Deep Recommender Architecture

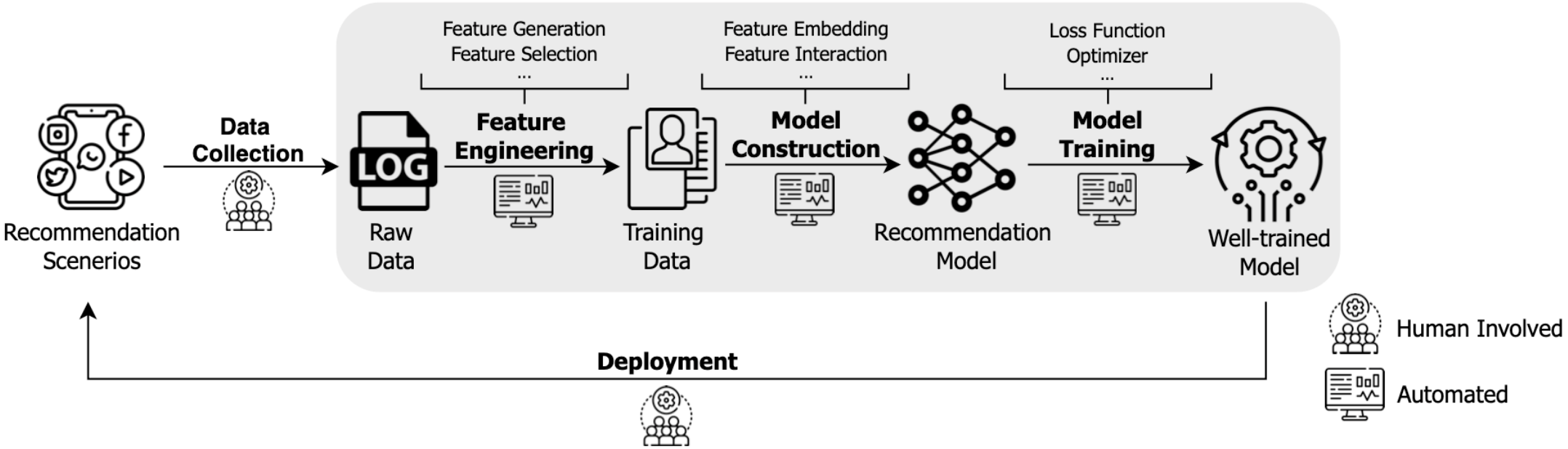


- **Advantages**
  - Feature representations of users and items
  - Non-linear relationships between users and items

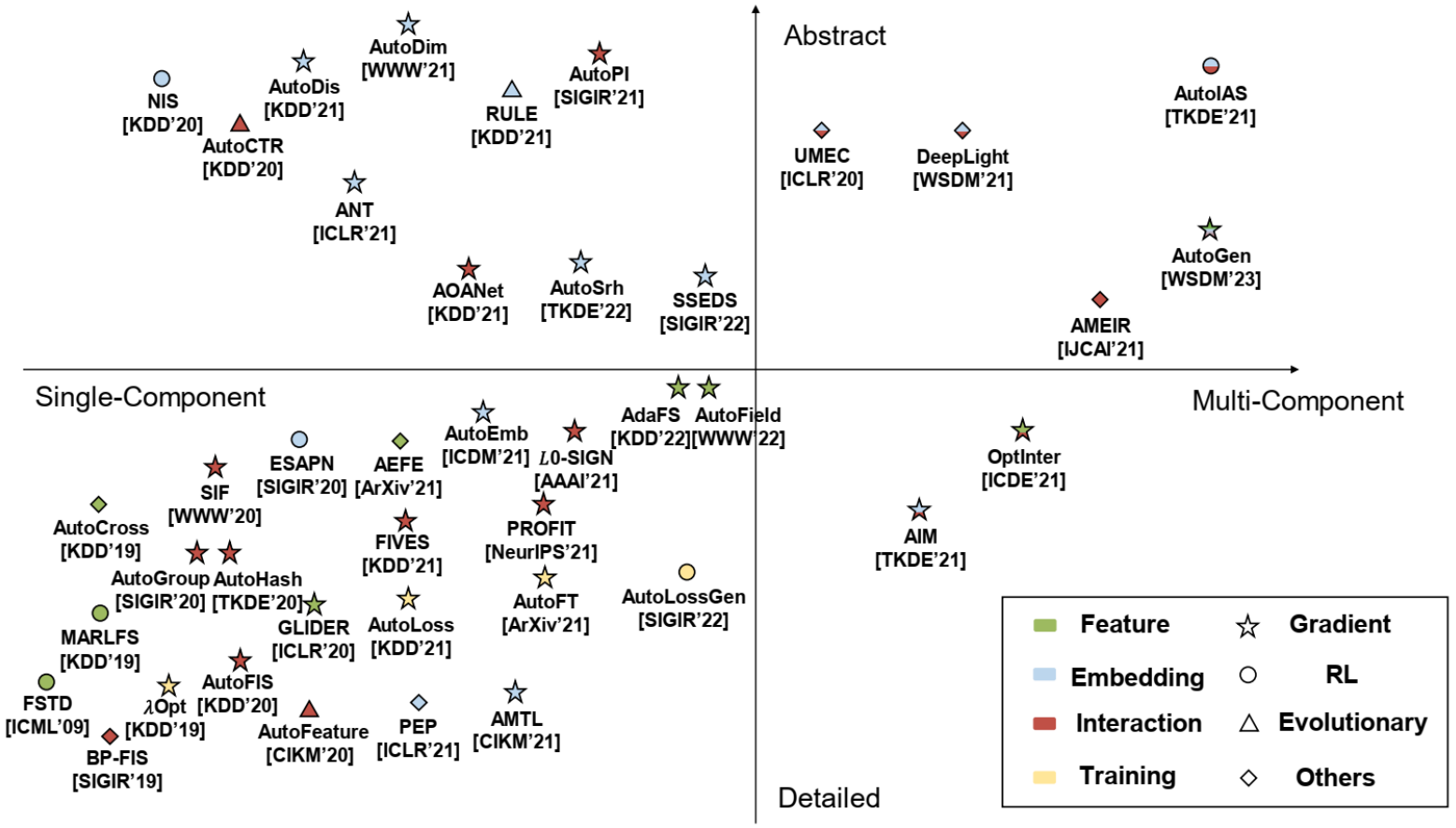
- **Manually designed architecture:**
  - Extensive expertise
  - Substantial engineering labor and time cost
  - Human bias and error

AutoML for Deep Recommender Systems

- Deep architectures are designed by the machine automatically
- Advantages
  - Less expert knowledge
  - Saving time and efforts
  - Different data -> different architectures



AutoML for Deep Recommender Systems



The trend of AutoML for recommender system

- Existing AutoML-based work evolves from **single-component** search to **multi-component** joint search.
- The search space of these AutoML-based work develops from **detailed** to **abstract** for shrinking search space and improving search efficiency.
- The search algorithm of existing work is mainly based on gradient-based methods, thus providing **efficient model searching and training mode**.

Automated Machine Learning for Recommendations

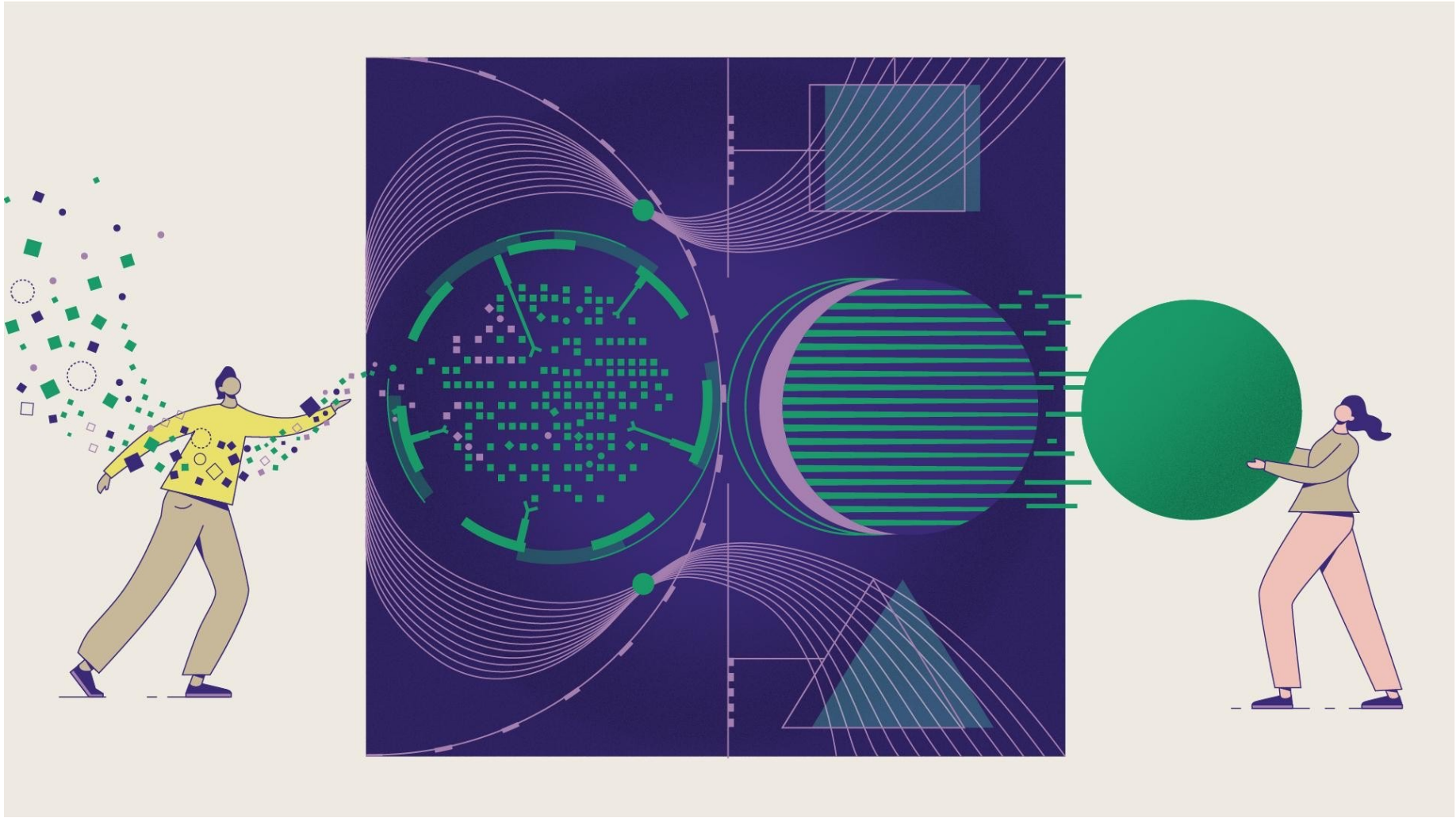
- **Agenda**
  - Introduction (Dr. LI Yong)
  - Preliminary of AutoML (Dr. ZHAO Xiangyu)
  - DRS Feature Selection (Mr. WANG Yejing)
  - DRS Embedding Component (Mr. CHEN Bo)
  - DRS Interaction Component (Dr. TANG Ruiming)
  - DRS Model Training (Dr. ZHAO Xiangyu)
  - DRS Comprehensive Search (Dr. ZHAO Xiangyu)
  - Conclusion & Future Direction (Dr. FAN Wenqi)
  - Q&A

A Comprehensive Survey on Automated Machine Learning for Recommendations. arXiv:2204.01390

Tutorial Website (Slides): https://advanced-recommender-systems.github.io/AutoML-Recommendations/

wsdm 2023 SINGAPORE FEBRUARY 27 - MARCH 3

Why AutoML?



Success of Machine Learning

Astronomy

Robotic

Teaching

Material Design

Creative Arts

Energy

Game Play

Search

Chemistry

Health Care

Physics

Image Recognition

Weather Prediction

Product Recommendation

Drug Discovery

Service

Financial Services

Manufacturing

Retail

Credit Assignment

Traffic Prediction

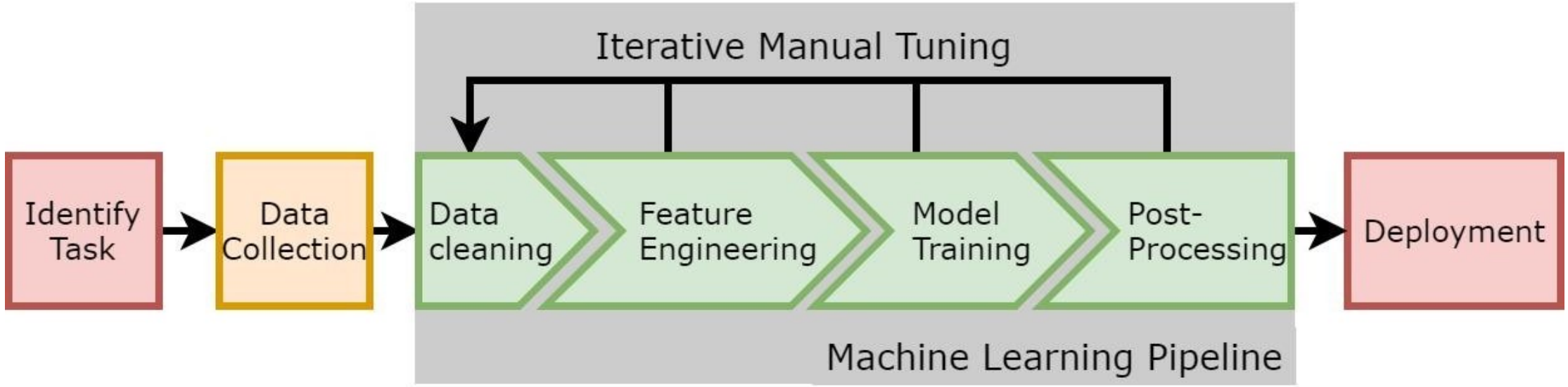
Media

Maintenance Prediction

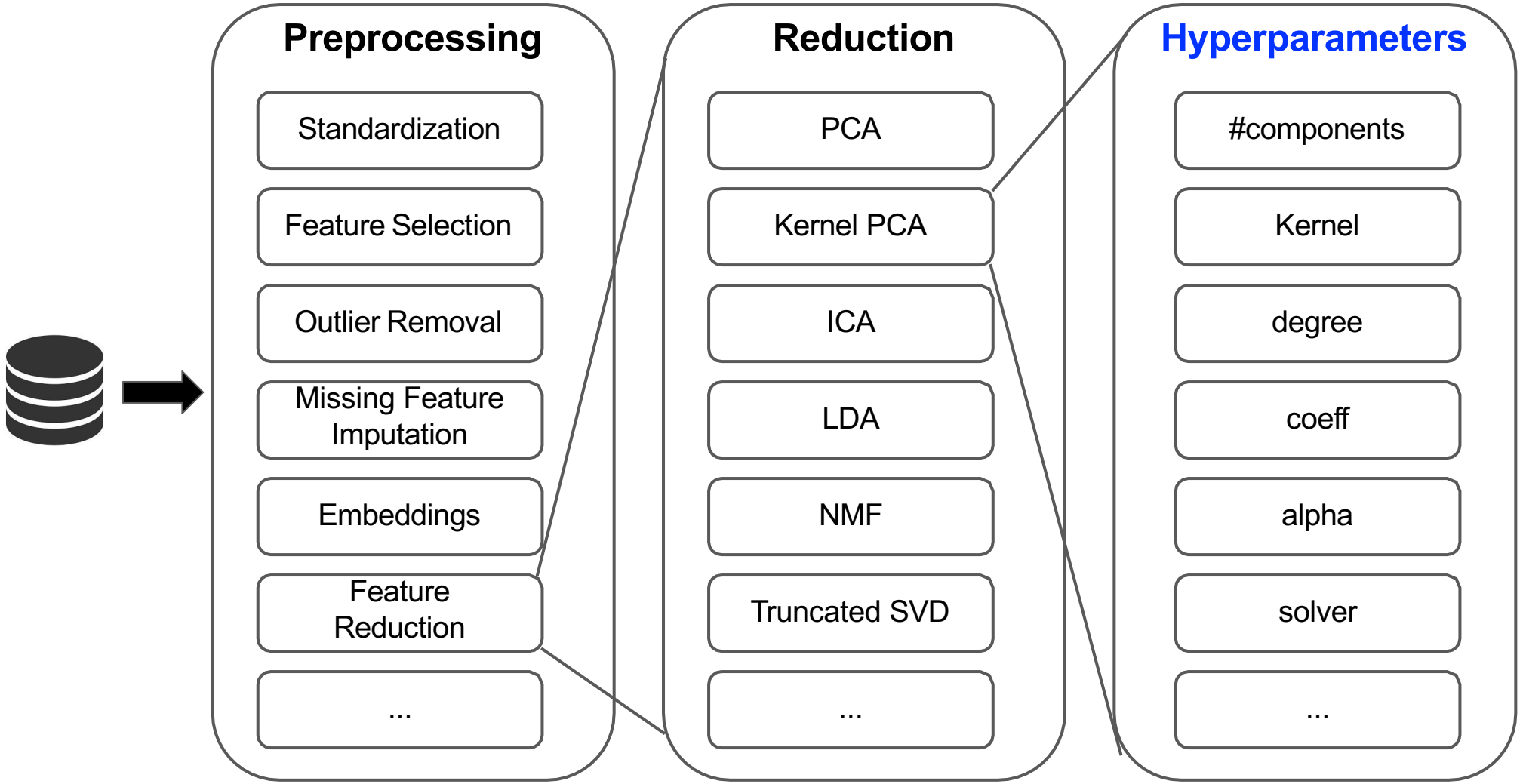
Social Media

Summary Generation

Machine Learning Pipeline

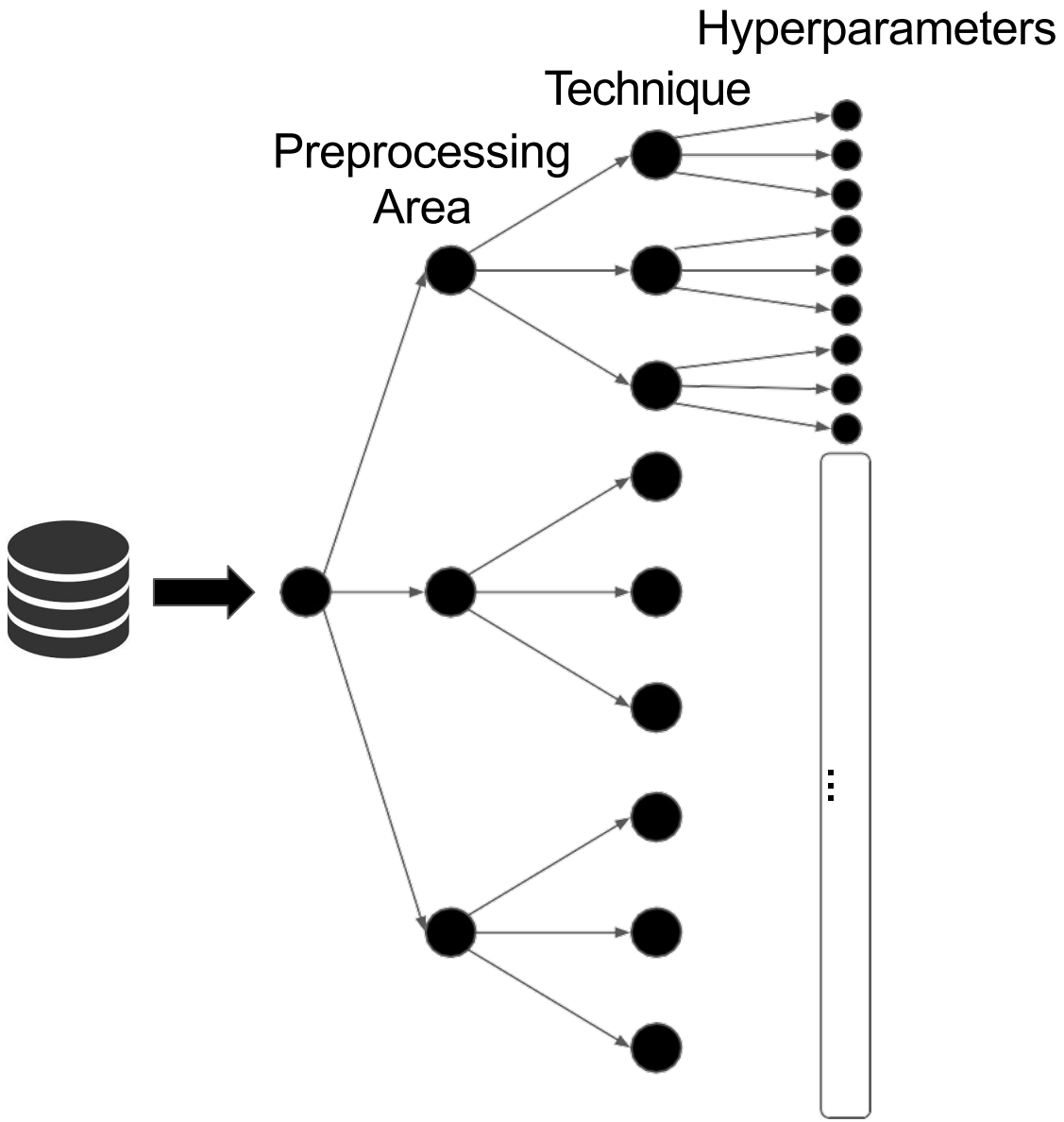


Algorithms and Hyperparameters



→ **We might want more than 1 data preprocessor!**

Complexity and the Best Combination



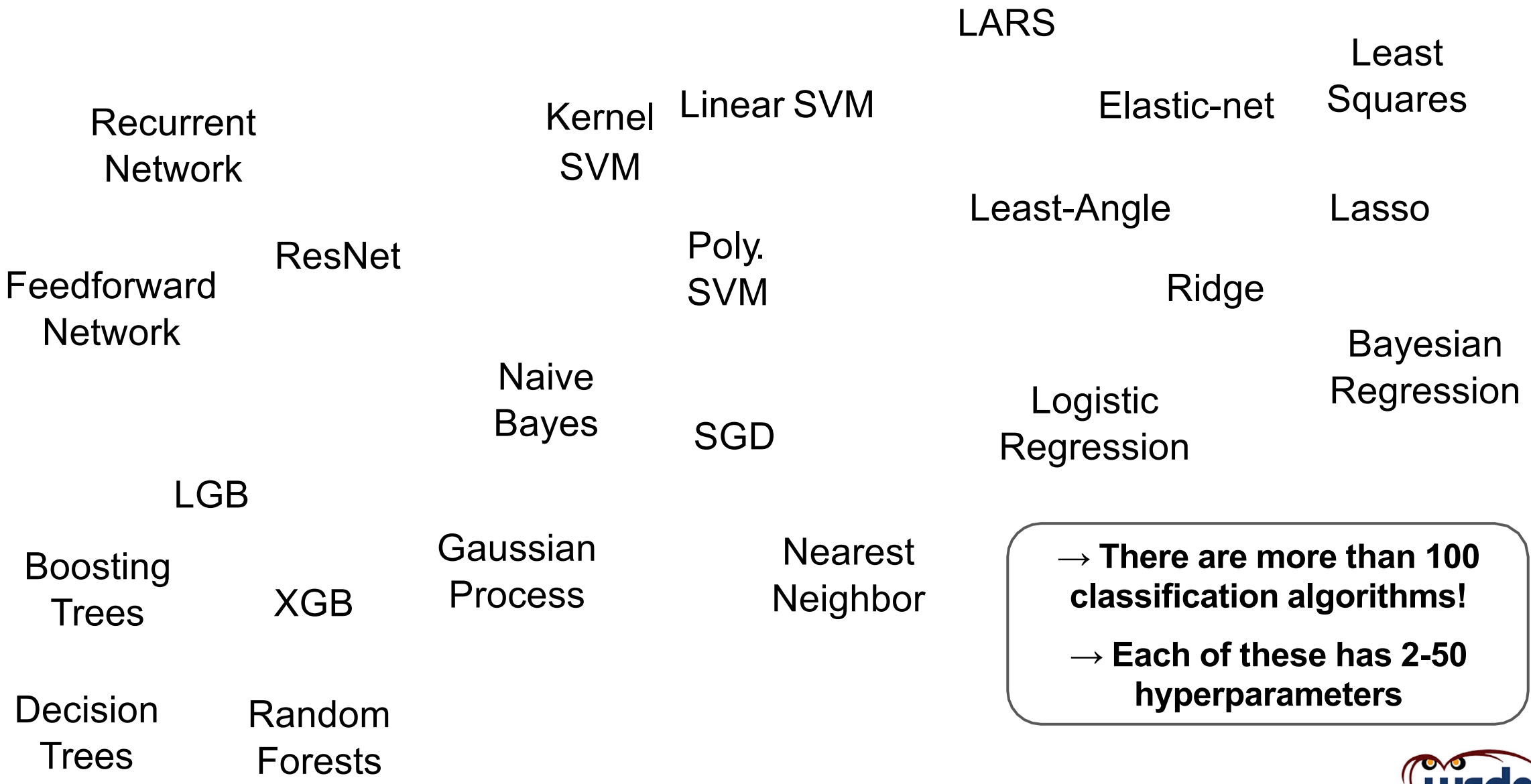
- Naive Assumptions:
  only 3 decisions at each level
- **Possible options**: 3 x 3 x 3 = **27**

- More realistic assumption:
  at least 10 decisions at leach level
- **Possible options:** 10 x 10 x 10 = **1000**

- Choose 3 preprocessors instead of 1
  → 1000 x 1000 x 1000 =
  1 000 000 000

- Still naive!
  → Hyperparameters are often continuous and not discrete
  → **infinite amount of settings!**

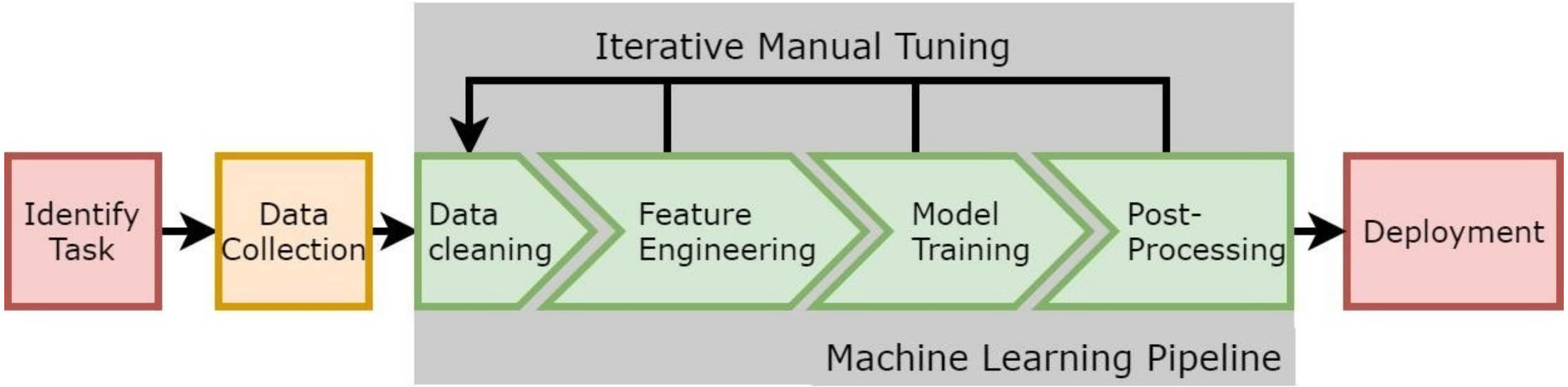
Classification Algorithms



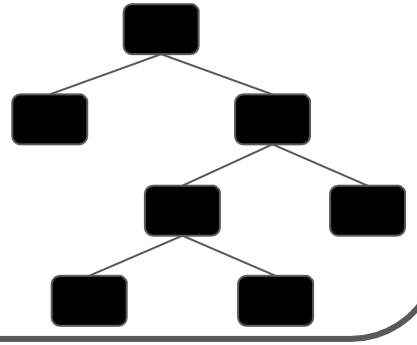
→ **There are more than 100 classification algorithms!**

→ **Each of these has 2-50 hyperparameters**

Challenges in Designing ML Pipelines



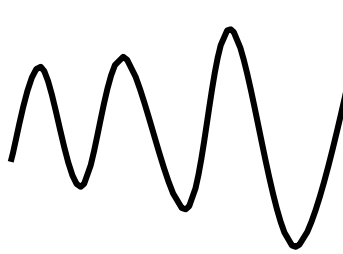
Complex Search Space



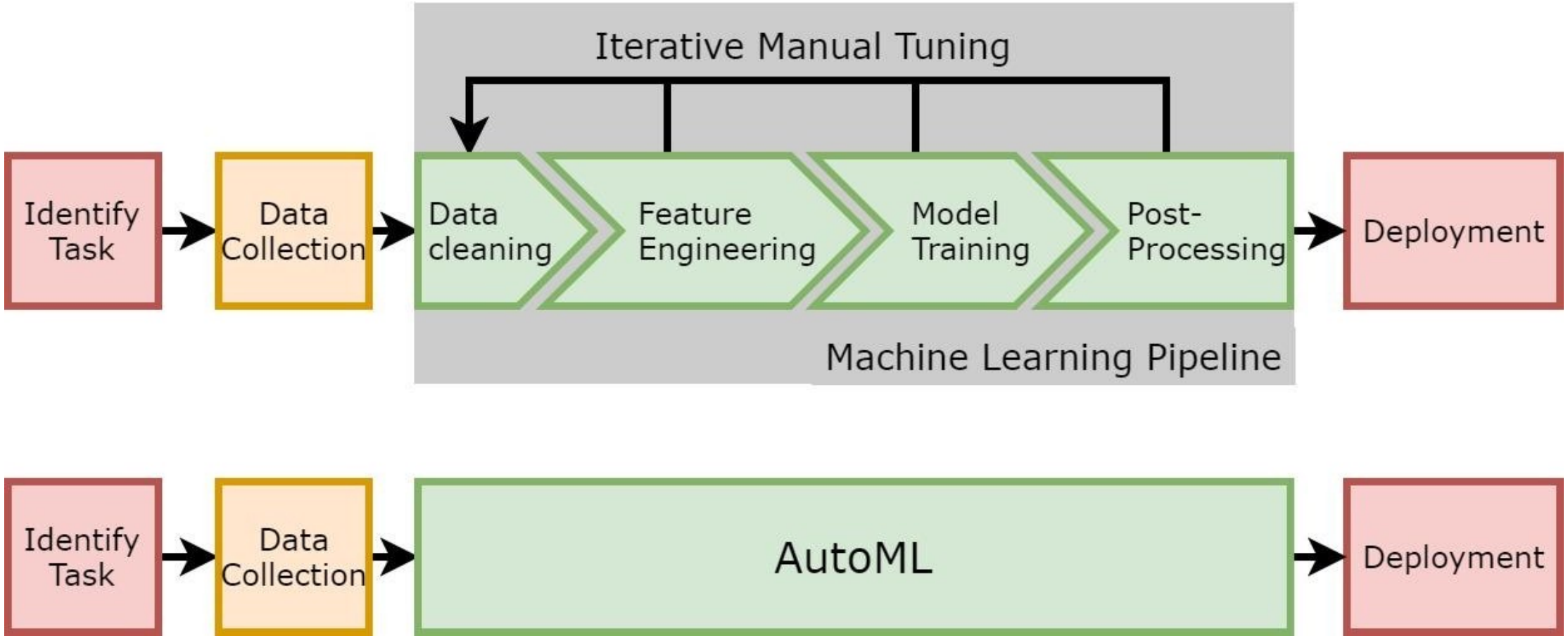
Black-Box Problem

Expensive Evaluations

Noise on observations



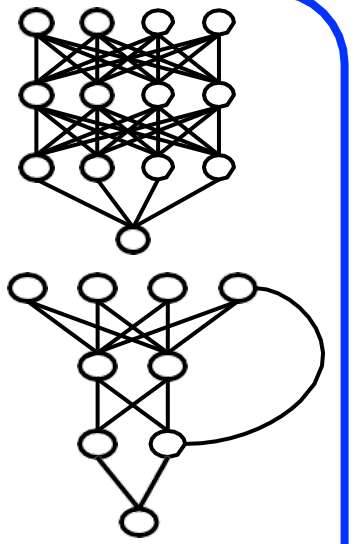
From Manual ML to Automated ML



Design Decisions by AutoML

classifier	#λ
AdaBoost (AB)	4
Bernoulli naïve Bayes	2
decision tree (DT)	4
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	4
linear SVM	4
kernel SVM	7
multinomial naïve Bayes	2
passive aggressive	3
QDA	2
random forest (RF)	5
Linear Class. (SGD)	10

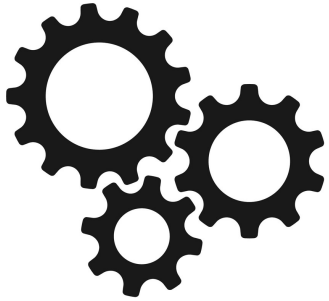
Algorithms



Architecture Design

preprocessor	#λ
extreml. rand. trees prepr.	5
fast ICA	4
feature agglomeration	4
kernel PCA	5
rand. kitchen sinks	2
linear SVM prepr.	3
no preprocessing	-
nystroem sampler	5
PCA	2
polynomial	3
random trees embed.	4
select percentile	2
select rates	3
one-hot encoding	2
imputation	1
balancing	1
rescaling	1

Pre-processing



Hyper-parameters

...

Neural Architecture Search (NAS)

- Find neural architecture A such that deep learning works best for given data
  - Measured by validation error of architecture A with trained weights $w^*(A)$

Neural Architecture Search (NAS)

- Find neural architecture A such that deep learning works best for given data
  - Measured by validation error of architecture A with trained weights $w^*(A)$

$$\min_{A \in \mathcal{A}} \mathcal{L}_{\text{val}}(w^*(A), A)$$

validation loss

$$\text{s.t. } w^*(A) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, A)$$

training loss

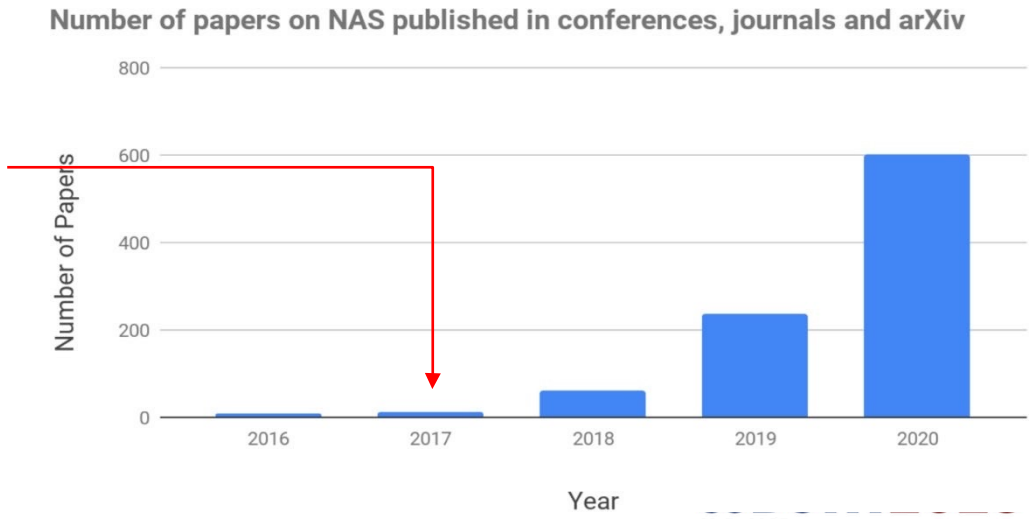
Neural Architecture Search (NAS)

- Find neural architecture A such that deep learning works best for given data
  - Measured by validation error of architecture A with trained weights $w^*(A)$

$$\min_{A \in \mathcal{A}} \mathcal{L}_{\text{val}}(w^*(A), A)$$

$$\text{s.t. } w^*(A) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, A)$$

- Famously tackled by reinforcement learning [Zoph & Le, ICLR 2017]
  - 12.800 architectures trained fully
  - 800 GPUs for 2 weeks (about $60.000 USD)

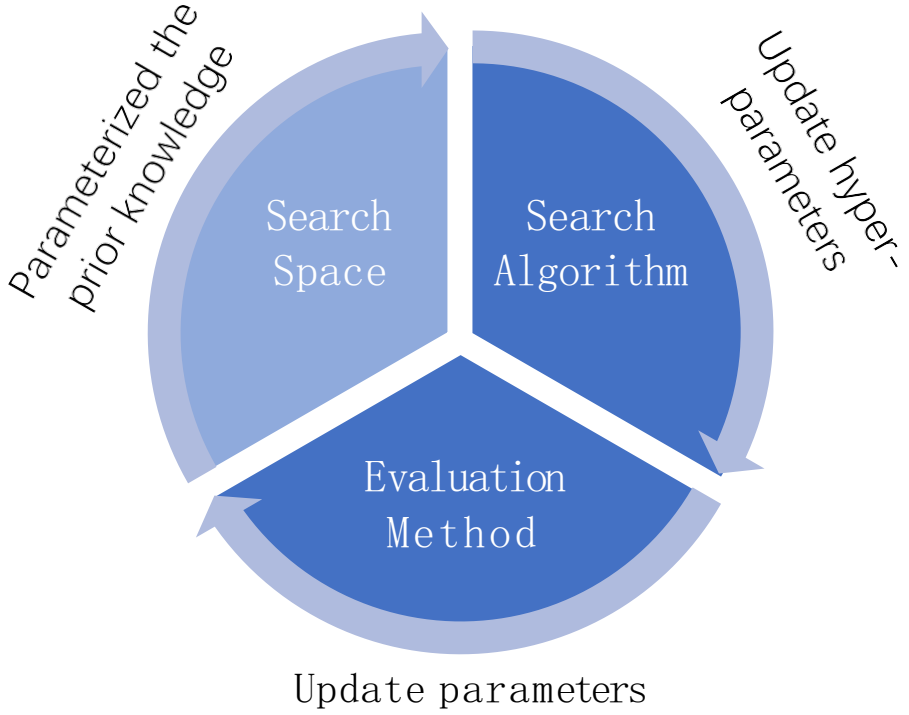


Major Components

- Search Space:
  - A set of operations (e.g. convolution, fully-connected, pooling)
  - how operations can be connected to form valid network architectures

operations

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv



Search Space

- Building blocks are like basic *genes* for these individuals
- Some examples here
  - Genetic CNN: only 3×3 convolution is allowed to be searched (followed by default BN and ReLU operations), 3×3 pooling is fixed
  - NASNet: 13 operations shown below
  - PNASNet: 8 operations, removing those never-used ones from NASNet
  - ENASNet: 6 operations
  - DARTS: 8 operations

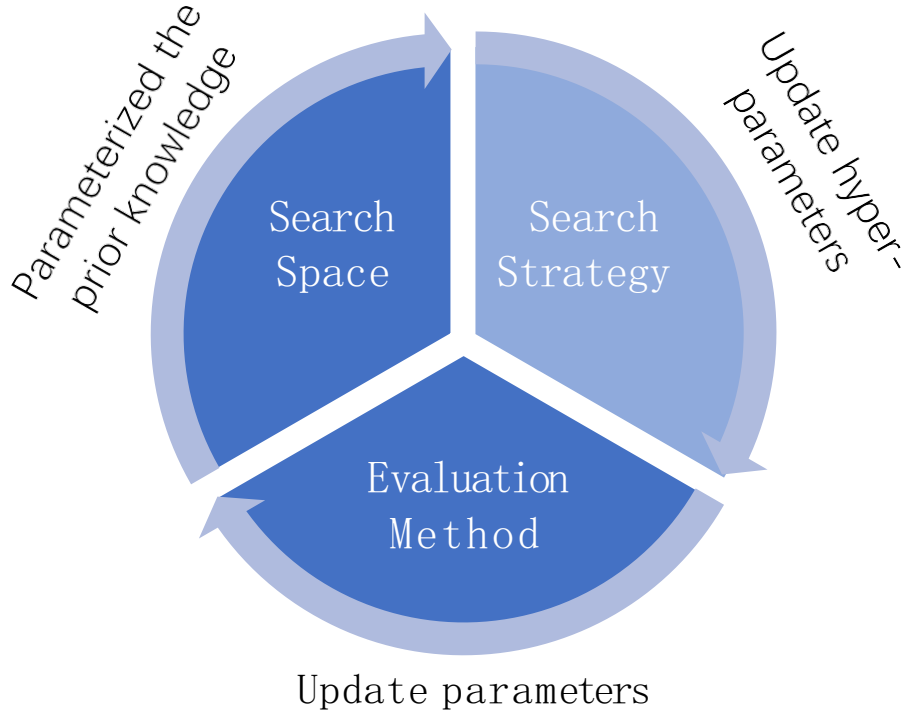
- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

[Xie, 2017] **L. Xie** *et al.*, Genetic CNN, *ICCV*, 2017.
[Zoph, 2018] **B. Zoph** *et al.*, Learning Transferable Architectures for Scalable Image Recognition, *CVPR*, 2018.
[Liu, 2018] **C. Liu** *et al.*, Progressive Neural Architecture Search, *ECCV*, 2018.
[Pham, 2018] **H. Pham** *et al.*, Efficient Neural Architecture Search via Parameter Sharing, *ICML*, 2018.
[Liu, 2019] **H. Liu** *et al.*, DARTS: Differentiable Architecture Search, *ICLR*, 2019.

Major Components

- Search Strategy
  - Sampling a population of network architecture candidates (child models)
  - Rewards: child model performance metrics (e.g. high accuracy, low latency)

- Algorithms
  - Random Search
  - Reinforcement Learning
  - Gradient descent
  - Evolutionary Algorithms



Search Algorithm

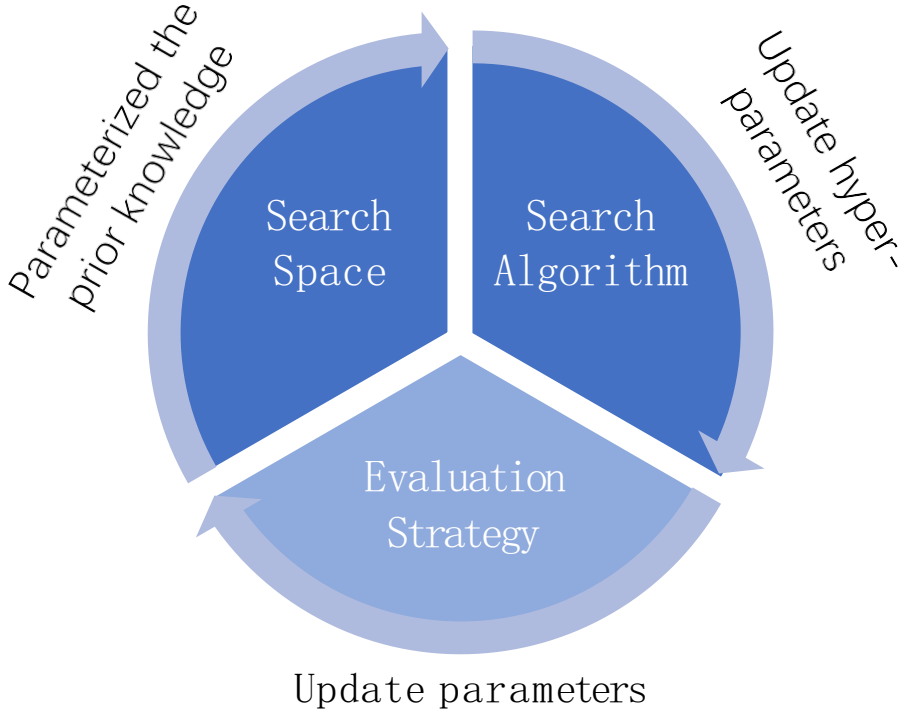
- Finding new individuals that have potentials to work better
  - Heuristic search in the large space
- Two mainly applied methods: the genetic algorithm and reinforcement learning
  - Both are heuristic algorithms applied to the scenarios of a large search space and limited ability to explore every single element in the space
  - A fundamental assumption: both of these heuristic algorithms can preserve good genes and based on which discover possible improvements
- Also, it is possible to integrate architecture search to network optimization
  - These algorithms are often much faster

[Real, 2017] **E. Real** *et al.*, Large-Scale Evolution of Image Classifiers, *ICML*, 2017.
[Xie, 2017] **L. Xie** *et al.*, Genetic CNN, *ICCV*, 2017.
[Zoph, 2018] **B. Zoph** *et al.*, Learning Transferable Architectures for Scalable Image Recognition, *CVPR*, 2018.
[Liu, 2018] **C. Liu** *et al.*, Progressive Neural Architecture Search, *ECCV*, 2018.
[Pham, 2018] **H. Pham** *et al.*, Efficient Neural Architecture Search via Parameter Sharing, *ICML*, 2018.
[Liu, 2019] **H. Liu** *et al.*, DARTS: Differentiable Architecture Search, *ICLR*, 2019.

Major Components

- Evaluation Strategy
  - We need to estimate or predict the performance of child models
  - In order to obtain feedback for the search algorithm to learn

- Methods
  - Training from Scratch
  - Proxy Task Performance
  - Parameter Sharing
  - Prediction-Based



Evaluation Method

- Evaluation aims at determining which individuals are good and to be preserved
- Conventionally, this was often done by training a network from scratch
  - This is extremely time-consuming, so researchers often train NAS on a small dataset like CIFAR and then transfer the found architecture to larger datasets like ImageNet
  - Even in this way, the training process is really slow: Genetic-CNN requires 17 GPU-days for a single training process, and NAS-RL requires more than 20,000 GPU-days
- Efficient methods were proposed later
  - Ideas include parameter sharing (without the need of re-training everything for each new individual) and using a differentiable architecture (joint optimization)
  - Now, an efficient search process on CIFAR can be reduced to a few GPU-hours, though training the searched architecture on ImageNet is still time-consuming

[Xie, 2017] **L. Xie** *et al.*, Genetic CNN, *ICCV*, 2017.
[Zoph, 2017] **B. Zoph** *et al.*, Neural Architecture Search with Reinforcement Learning, *ICLR*, 2017.
[Pham, 2018] **H. Pham** *et al.*, Efficient Neural Architecture Search via Parameter Sharing, *ICML*, 2018.
[Liu, 2019] **H. Liu** *et al.*, DARTS: Differentiable Architecture Search, *ICLR*, 2019.

NAS with Reinforcement Learning

- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank

NAS with Reinforcement Learning

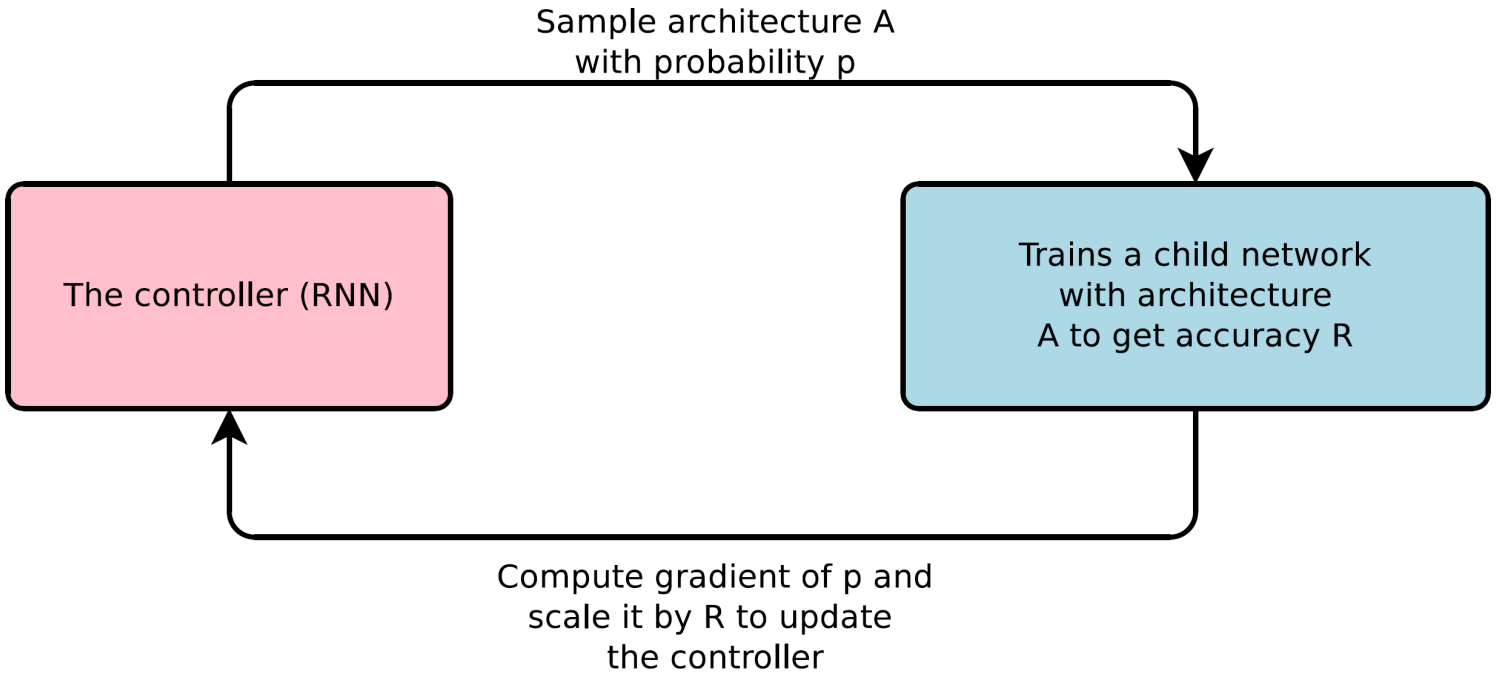
- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands:

    800 GPUs for 3-4 weeks, 12.800 architectures trained

NAS with Reinforcement Learning

- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands:

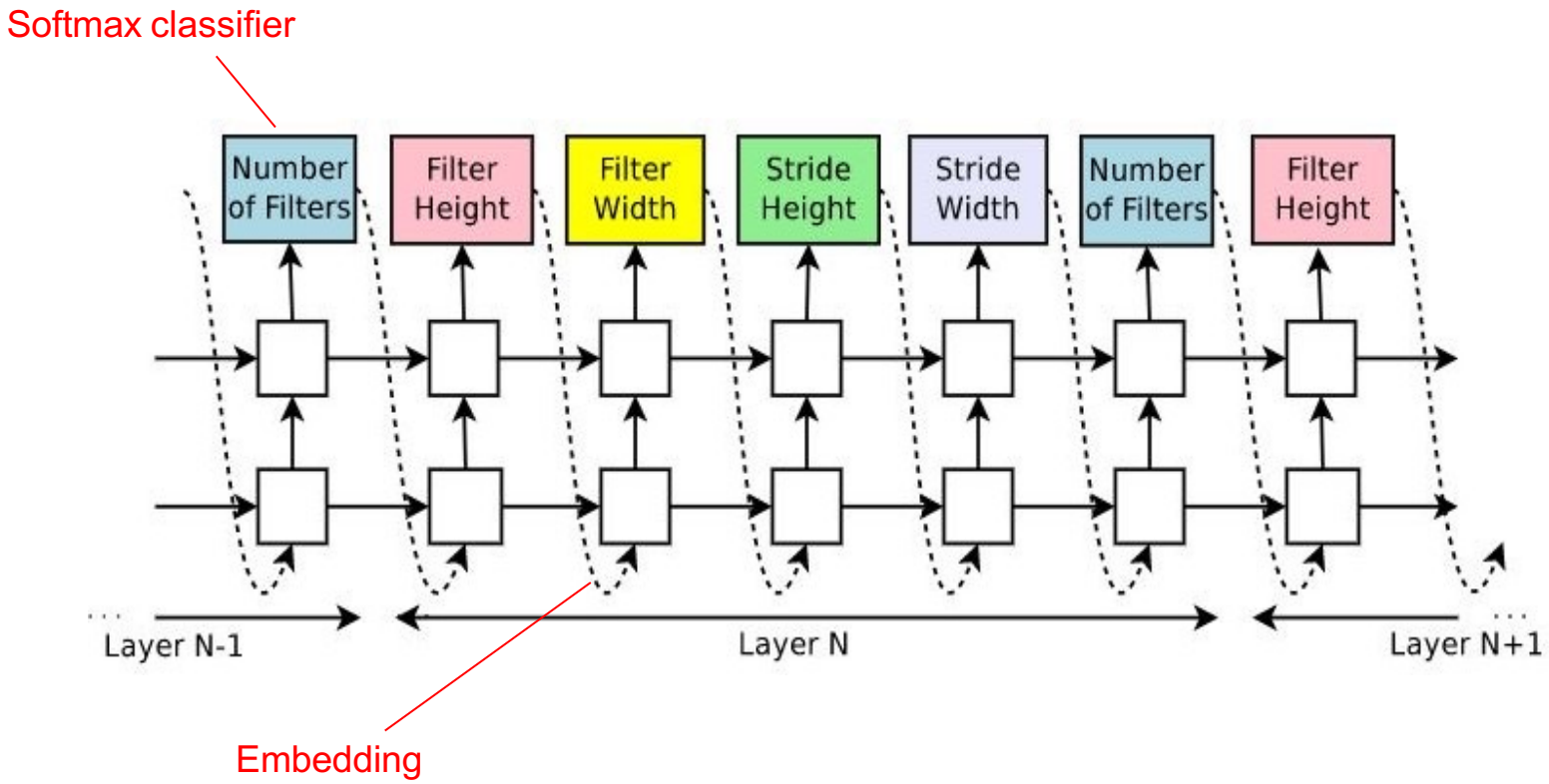
    800 GPUs for 3-4 weeks, 12.800 architectures trained



NAS with Reinforcement Learning

[Zoph & Le, ICLR 2017]

- Architecture of neural network represented as string e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”]
- Controller (RNN) generates string that represents architecture



Training with REINFORCE

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

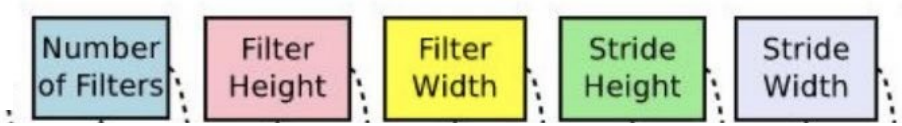
$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN viewed as a sequence of actions

NAS as Hyperparameter Optimization

[Zoph & Le, ICLR 2017]

- Architecture of neural network represented as string e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”]
- We can simply treat these as categorical parameters
  - E.g., 25 cat. parameters for each of the 2 cells in [Zoph et al, CVPR 2018]



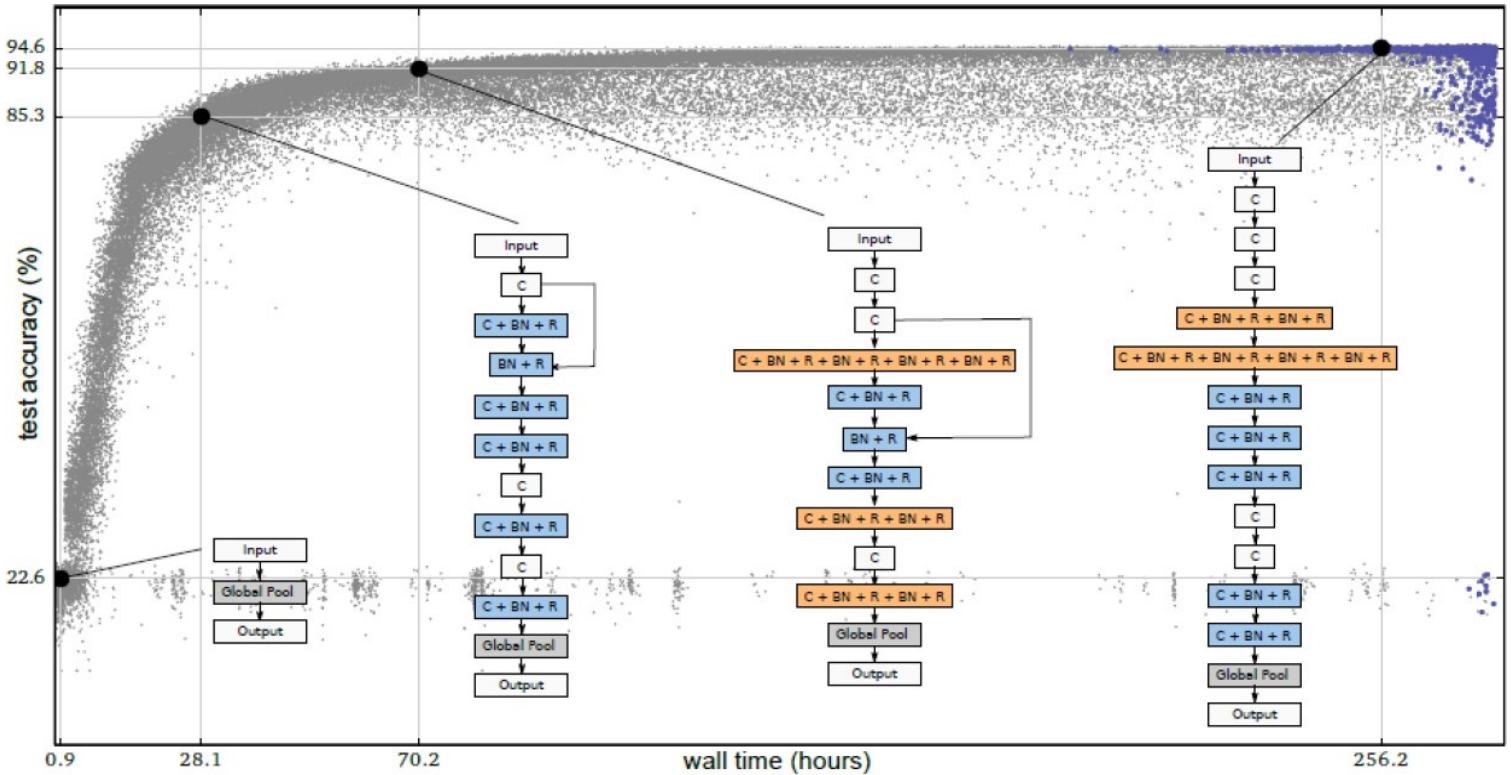
NAS with Evolution

- Neuroevolution

  (already since the 1990s [Angeline et al., 1994; Stanley and Miikkulainen, 2002])

  - Mutation steps, such as adding, changing or removing a layer

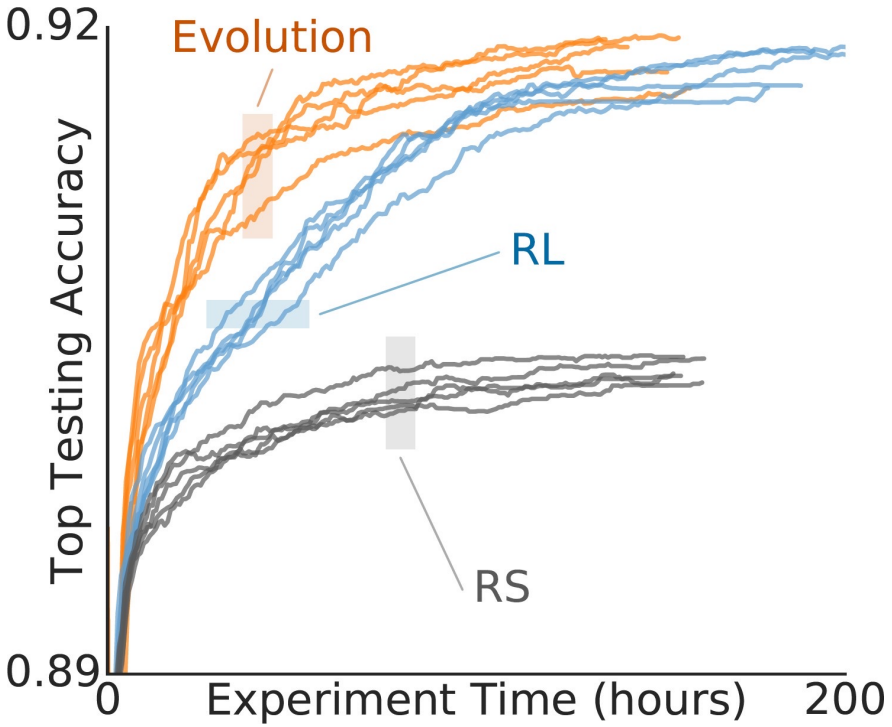
    [Real et al., ICML 2017; Miikkulainen et al., arXiv 2017]



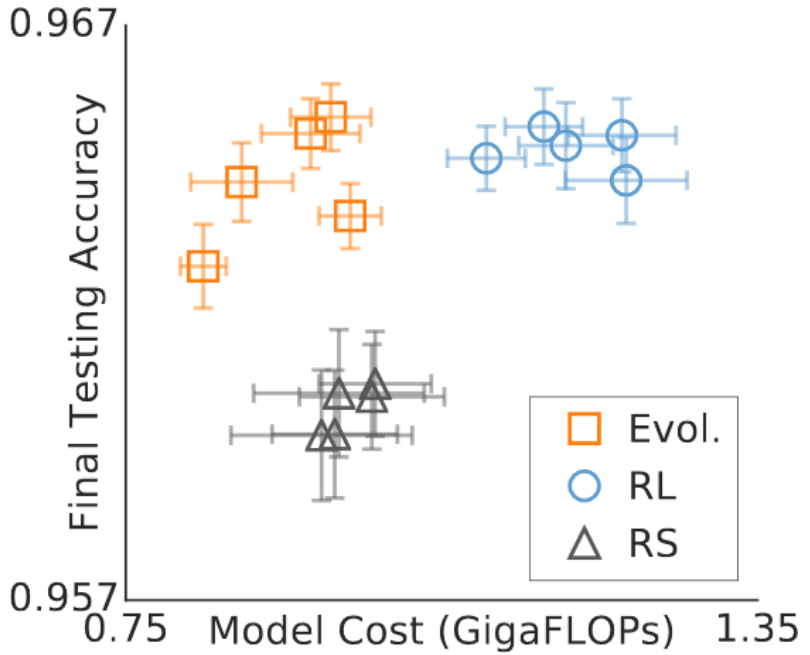
RL vs. Evolution vs. Random Search

[Real et al., AAAI 2019]

during architecture search



final evaluation



Huge Compute of Blackbox Methods

Dataset:
CIFAR-10

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Zoph and Le (2017)	3.65	37.4	22,400
RL	Zoph et al. (2018)	3.41	3.3	2,000
EA	Real et al. (2017)	5.40	5.4	2,600
EA	Real et al. (2019)	3.34	3.2	3,150

Going to cell search space

[Wistuba et al., preprint 2019]

Overview of NAS Speedup Techniques

- **Weight Inheritance & Network Morphisms**
  - Local changes in architecture, followed by fine-tuning steps
  - [Cai et al, 2018; Elsken et al, 2017; Cortes et al, 2017; Cai et al, 2018, Elsken et al, 2019]
- **Weight Sharing & One-Shot Models**
  - ENAS [Pham et al, 2018], DARTS [Liu et al, 2019] and many follow-ups
- **Meta-Learning**
  - Learning across datasets
  - To initialize architectural weights of DARTS [Lian et al, 2020; Elsken et al, 2020]
  - Prior for blackbox optimization methods [Wong et al, 2018; Runge et al, 2019; Zimmer et al, 2020]
- **Multi-Fidelity Optimization**
  - Exploit cheaper proxy models for blackbox optimizers, in particular Bayesian optimization
  - [Jamieson & Talwalkar, 2016; Li et al, 2017; Falkner et al, 2018; Zela et al, 2018; White et al, 2021]

Network Morphisms

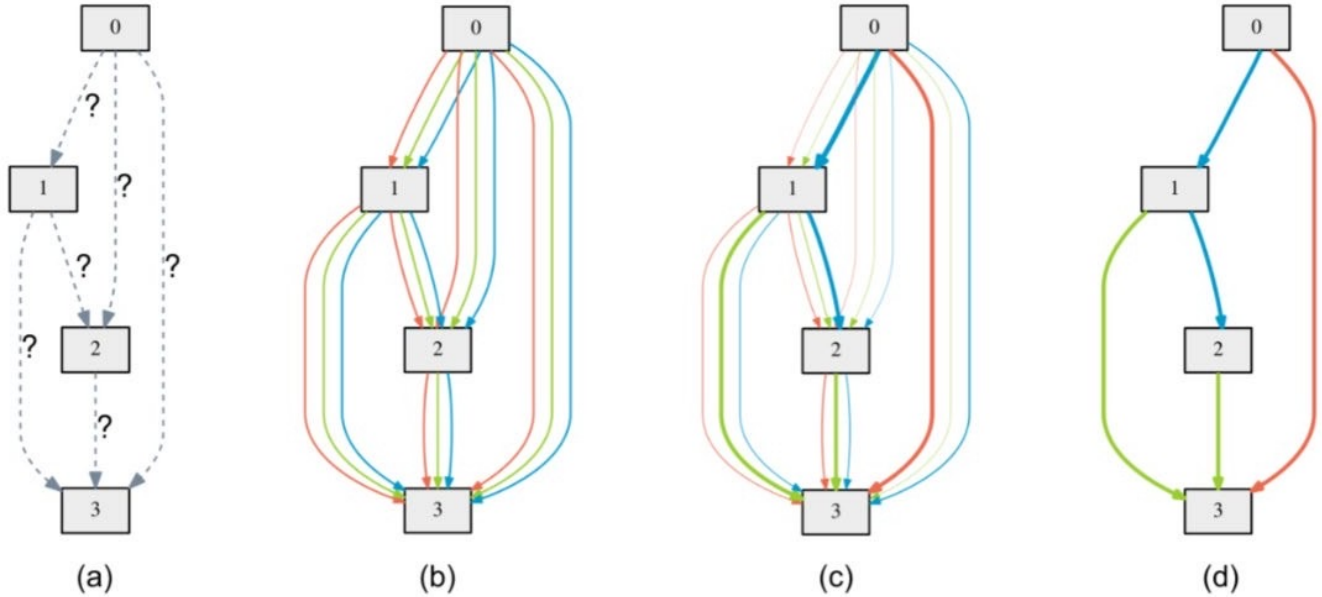
- Network morphisms [Chen et al., 2016; Wei et al., 2016]
  - Change the network structure, but not the modelled function (i.e., for every input, the network yields the same output
  - as before applying the network morphism)

  - Can use this in NAS algorithms as operations to generate new networks
  - Avoids costly training from scratch

Overview of NAS Speedup Techniques

- **Weight Inheritance & Network Morphisms**
  - Local changes in architecture, followed by fine-tuning steps
  - [Cai et al, 2018; Elsken et al, 2017; Cortes et al, 2017; Cai et al, 2018, Elsken et al, 2019]
- **Weight Sharing & One-Shot Models**
  - ENAS [Pham et al, 2018], DARTS [Liu et al, 2019] and many follow-ups
- **Meta-Learning**
  - Learning across datasets
  - To initialize architectural weights of DARTS [Lian et al, 2020; Elsken et al, 2020]
  - Prior for blackbox optimization methods [Wong et al, 2018; Runge et al, 2019; Zimmer et al, 2020]
- **Multi-Fidelity Optimization**
  - Exploit cheaper proxy models for blackbox optimizers, in particular Bayesian optimization
  - [Jamieson & Talwalkar, 2016; Li et al, 2017; Falkner et al, 2018; Zela et al, 2018; White et al, 2021]

DARTS: Differentiable Architecture Search

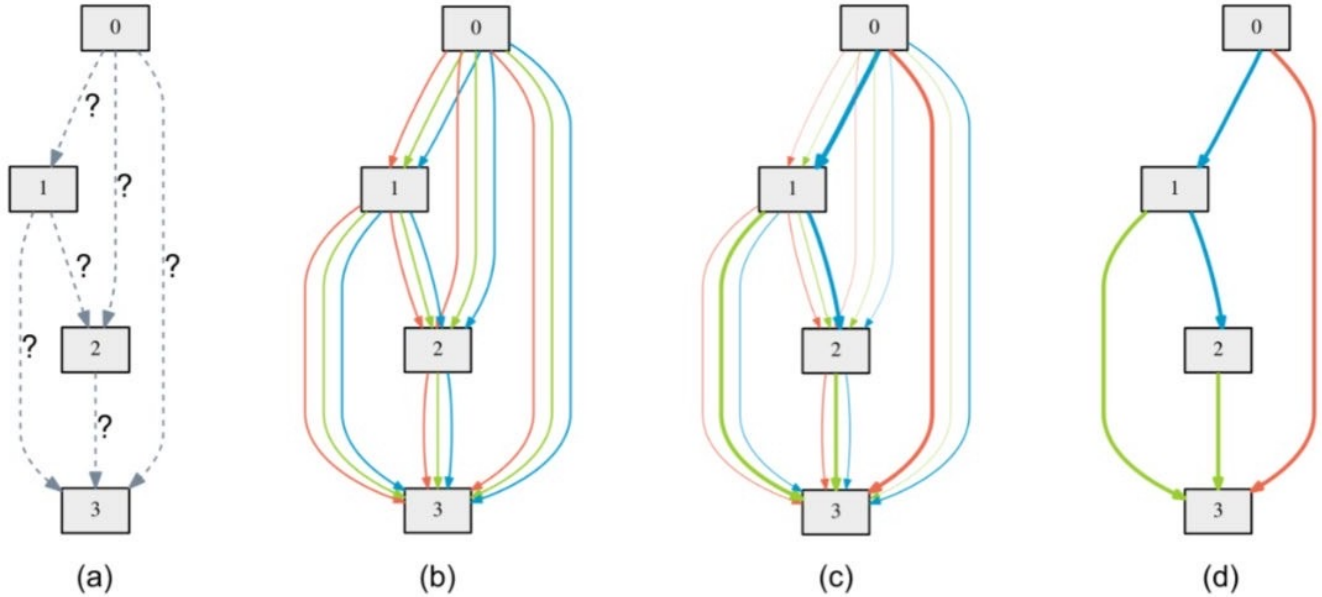


[Liu et al at ICLR 2019]

Candidate operations

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

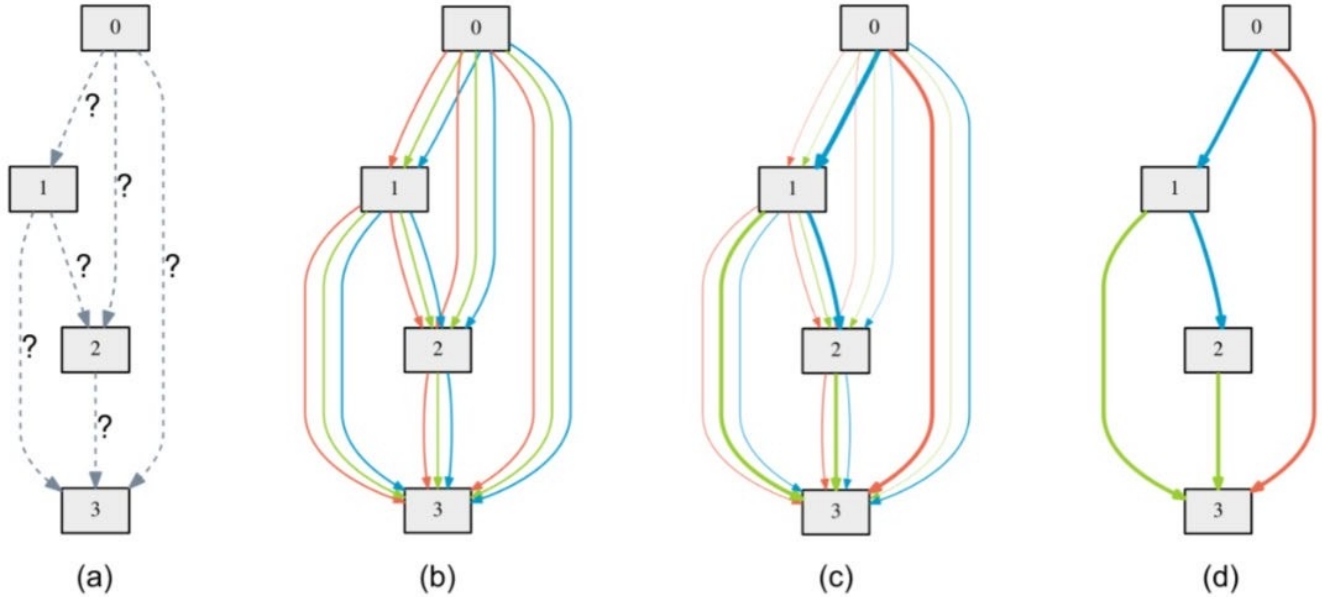
DARTS: Differentiable Architecture Search



[Liu et al at ICLR 2019]

- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - Mixed operator: $\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$

DARTS: Differentiable Architecture Search



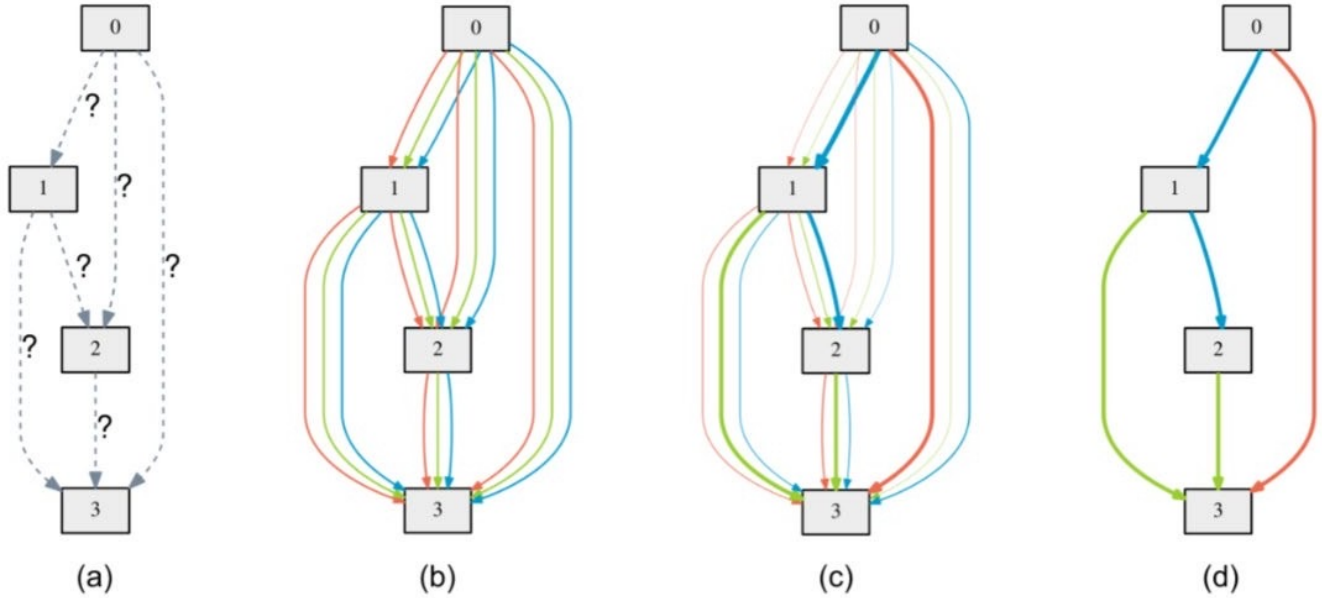
[Liu et al at ICLR 2019]

- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - Mixed operator: $\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$
- Solve a bi-level optimization problem (c)

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = \operatorname{argmin}_w \ \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

DARTS: Differentiable Architecture Search

[Liu et al at ICLR 2019]



- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - Mixed operator: $\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$
- Solve a bi-level optimization problem (c)

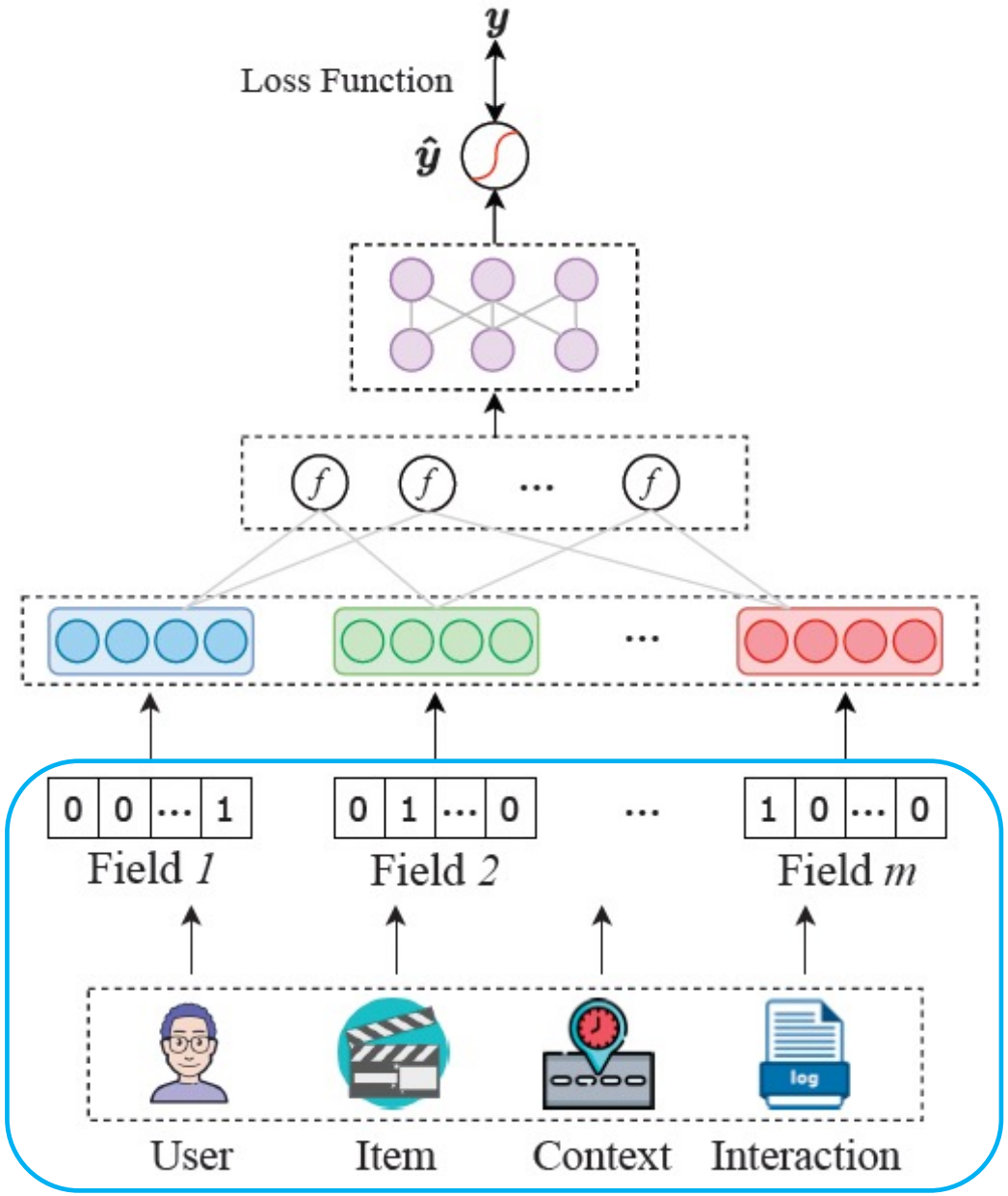
$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

- In the end, discretize to obtain a single architecture (d)

Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
  - Selection from raw features
  - Selection from generated features
- DRS Embedding Components
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

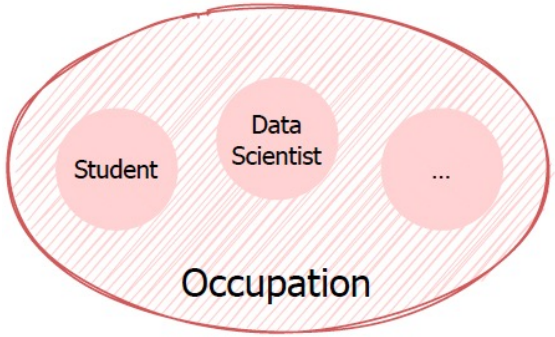
Background

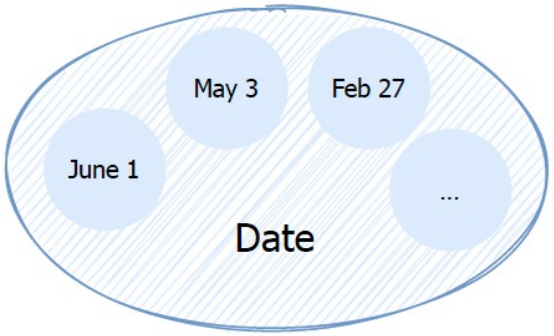


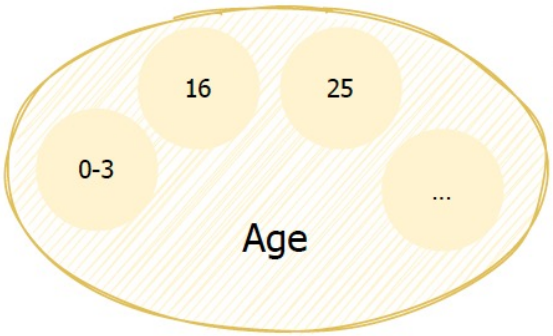
- Feature selection:
  - Select predictive features prior to model construction
- Importance:
  - Time/Memory efficiency
  - Accuracy
- Classification:
  - Candidates
  - Granularity
  - How they combine with DRS

Background – Candidates

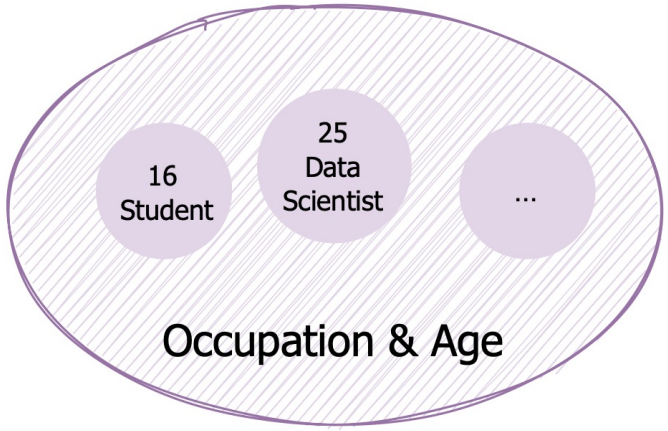
- Selection from raw features
  - Occupation
  - Age
  - Date
  - …

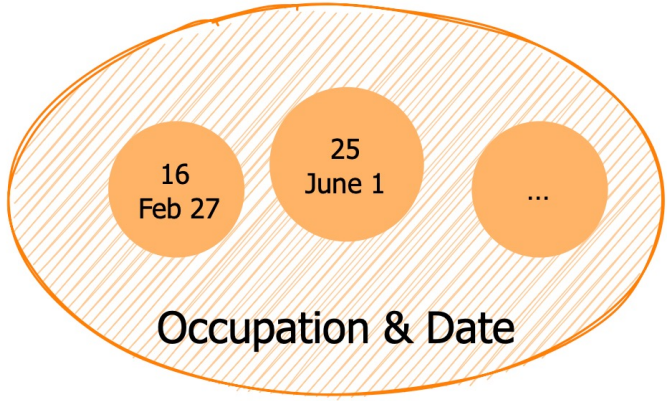






- Selection from generated features
  - Occupation & Age
  - Date & Age
  - …

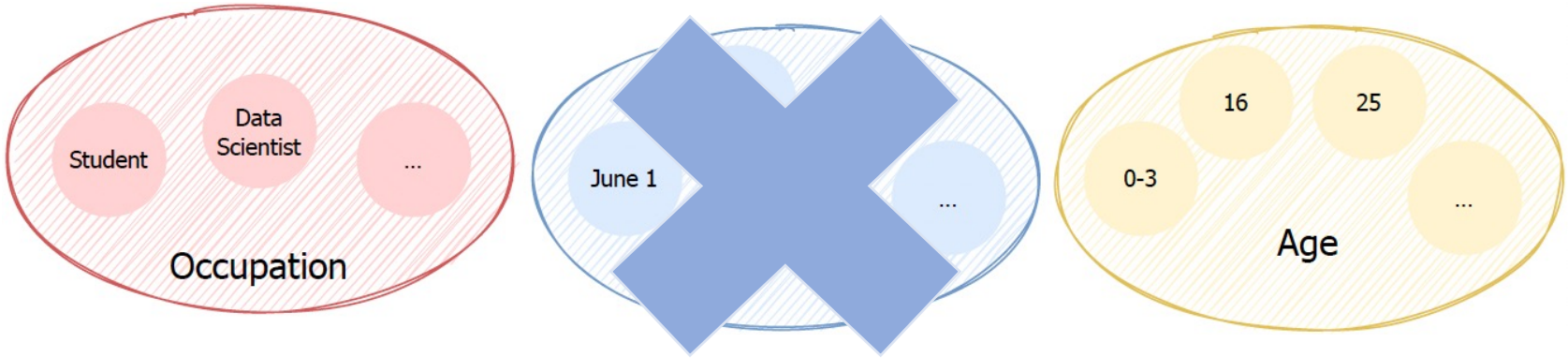




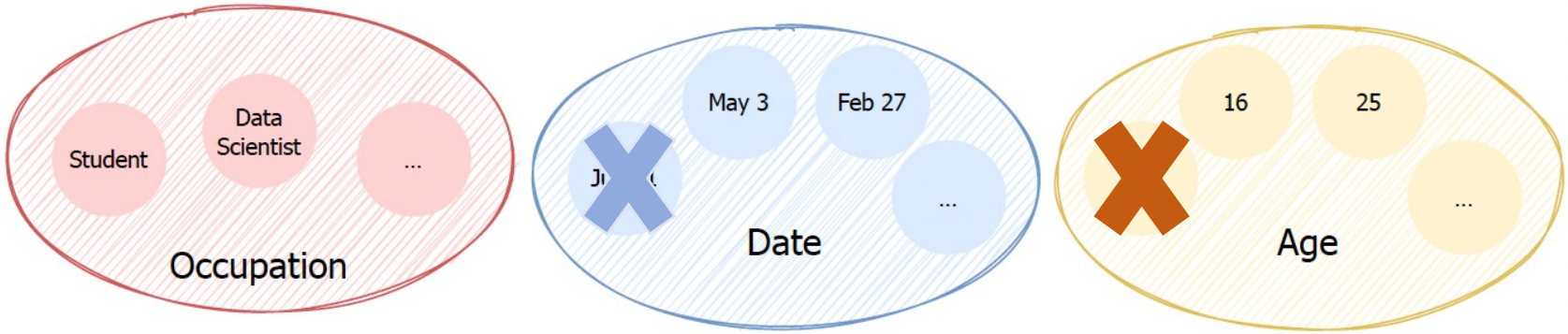
- Generated features v.s. Feature interaction
  - Explicit v.s. Implicit
  - Feature engineering v.s Model construction

Background – Granularity

- Field-level selection
  - Select or deselect the whole feature field
  - E.g., drop “Date”

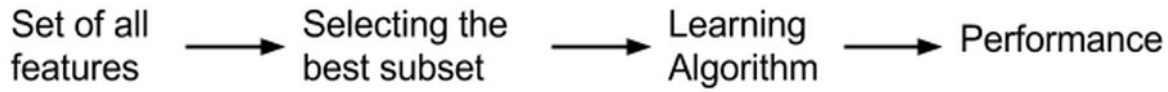


- Feature-level selection
  - Select or deselect specific feature values
  - E.g., drop “0-3” for “Age”

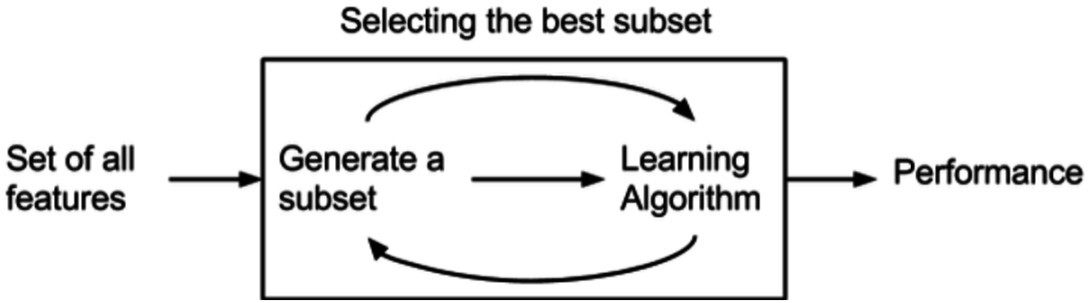


Background – Combination Methods

- Filter
  - Filter redundant features with specific scoring functions
  - Neglect dependency with subsequent DRS



- Wrapper
  - Elaborate feature set
  - Computationally intensive



- Embedded
  - Integrated into DRS
  - Fixed DRS

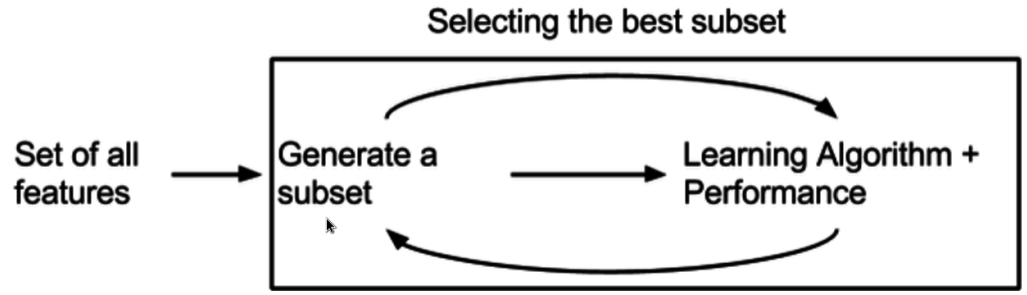


Table of Contents

- DRS Feature Selection
  - Selection from raw features
  - Selection from generated features

FSTD

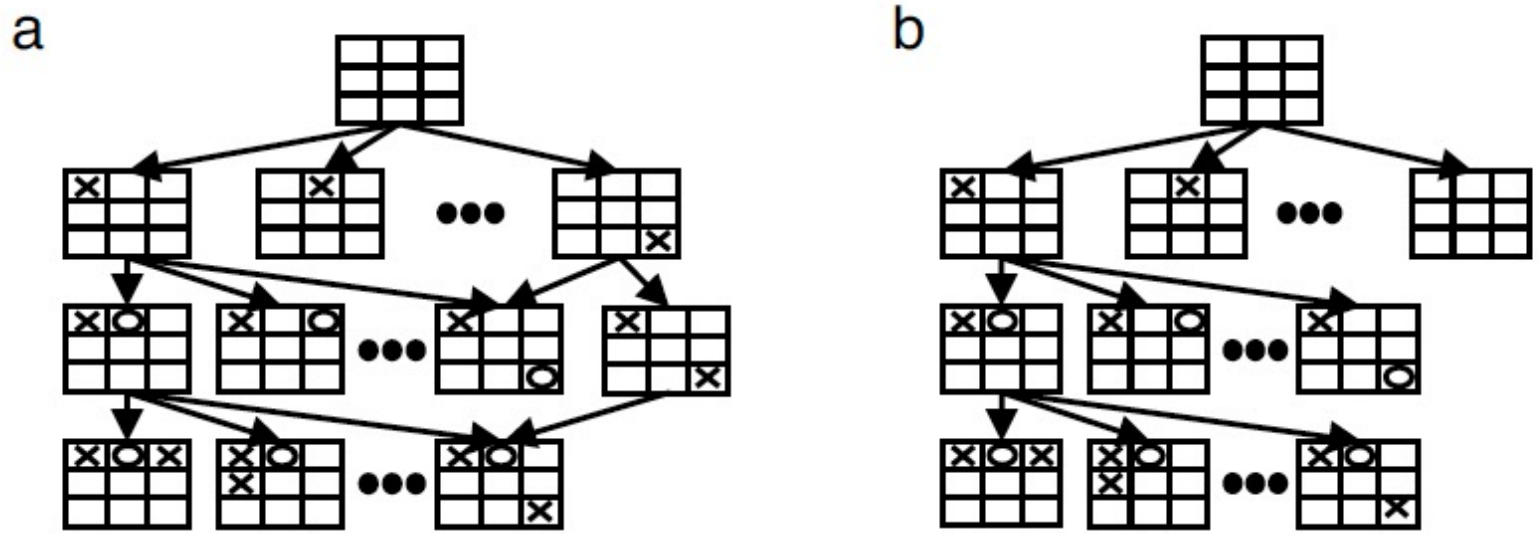
- Motivation:
  - Large search space of feature selection
  - Feature selection is a one-player game

- Target:
  - Select features using temporal difference (TD)
  - Field-level selection
  - Filter / Wrapper

Using reinforcement learning to find an optimal set of features, 2013

FSTD

- Strategy:
  - UCB-Phase (a. Exploit): Select the most predictive/stored feature
  - Random-Phase (b. Explore): Randomly select a feature
- Transition: Seen or unseen
- Reward: AUC score
- Optimization: Temporal difference + UCB



An example of Tic-Tac-Tau game.

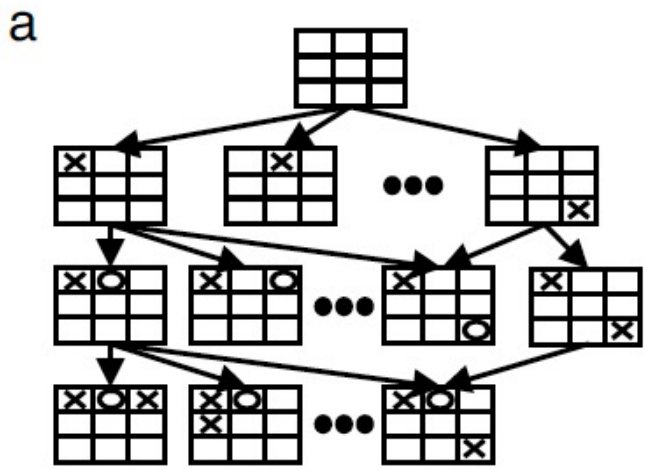
FSTD

- Scoring features: temporal difference (F: feature set, f: the new feature, V: AUC)

$$\mathrm{AOR}_f = \mathrm{Average}\{V(F_t) - V(F_{t+1})\}$$

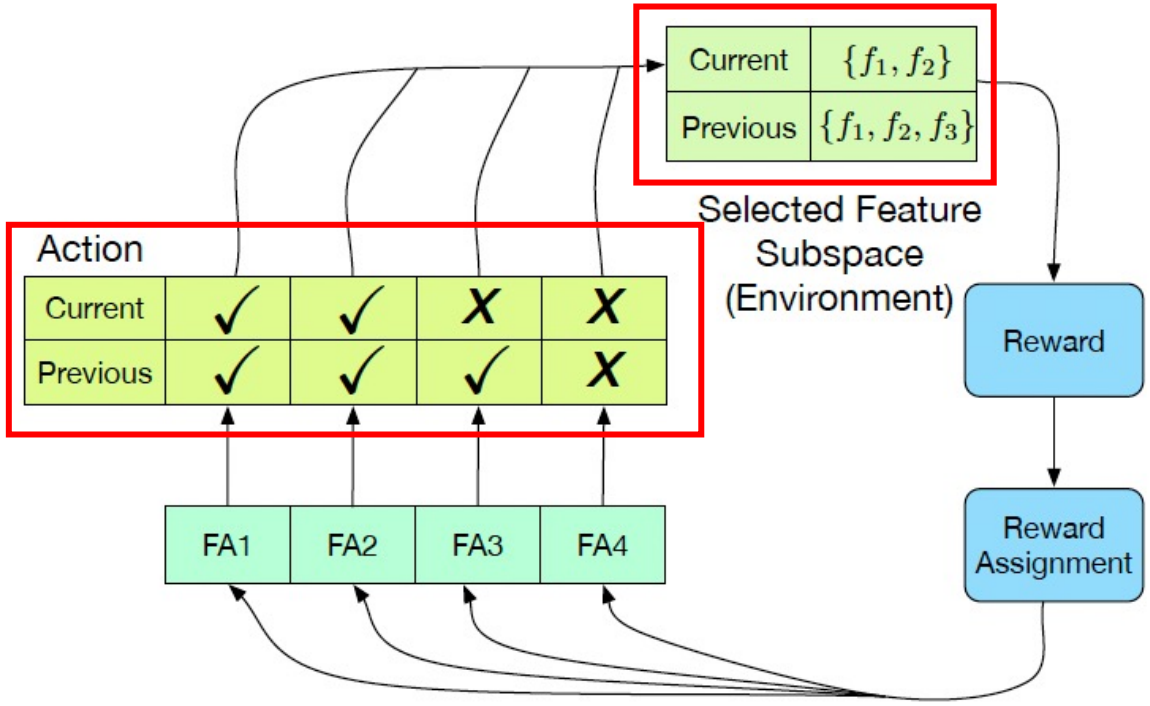
$$(\mathrm{AOR}_f)_{New} = [(k-1)(\mathrm{AOR}_f)_{Old} + V(F)]/k,$$

- Final selection:
  - F-FSTD: Select features with the highest AOR (single features)
  - W-FSTD: Search in the traversed graph space (feature sets)



MARLFS

- Target:
  - Select features via multi-agent RL
  - Field-level
  - Embedded

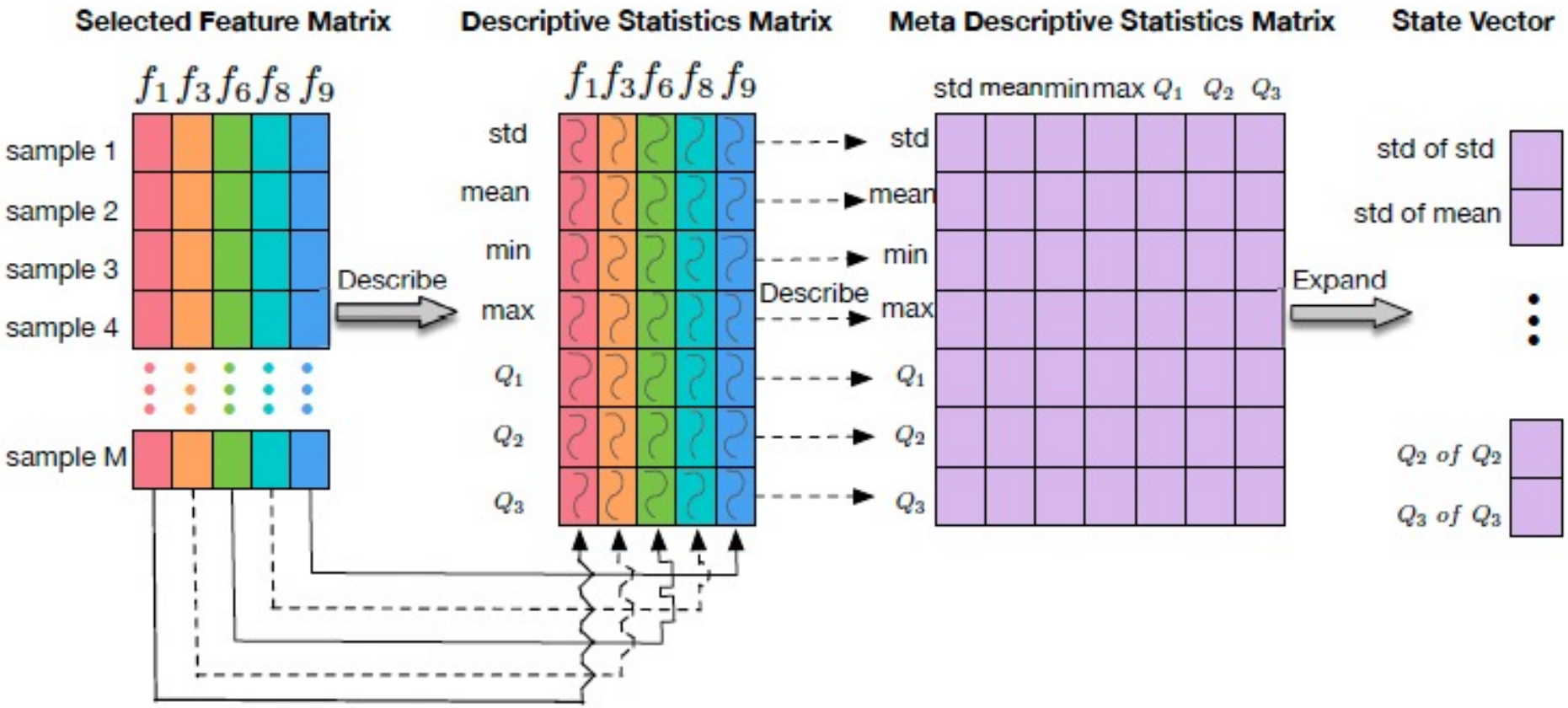


- Problem formulation:
  - Agent: N agents for N feature fields (1 on 1)
  - Action: Select or deselect the corresponding feature
  - Environment: Selected feature subset

Automating feature subspace exploration via multi-agent reinforcement learning, KDD 2019

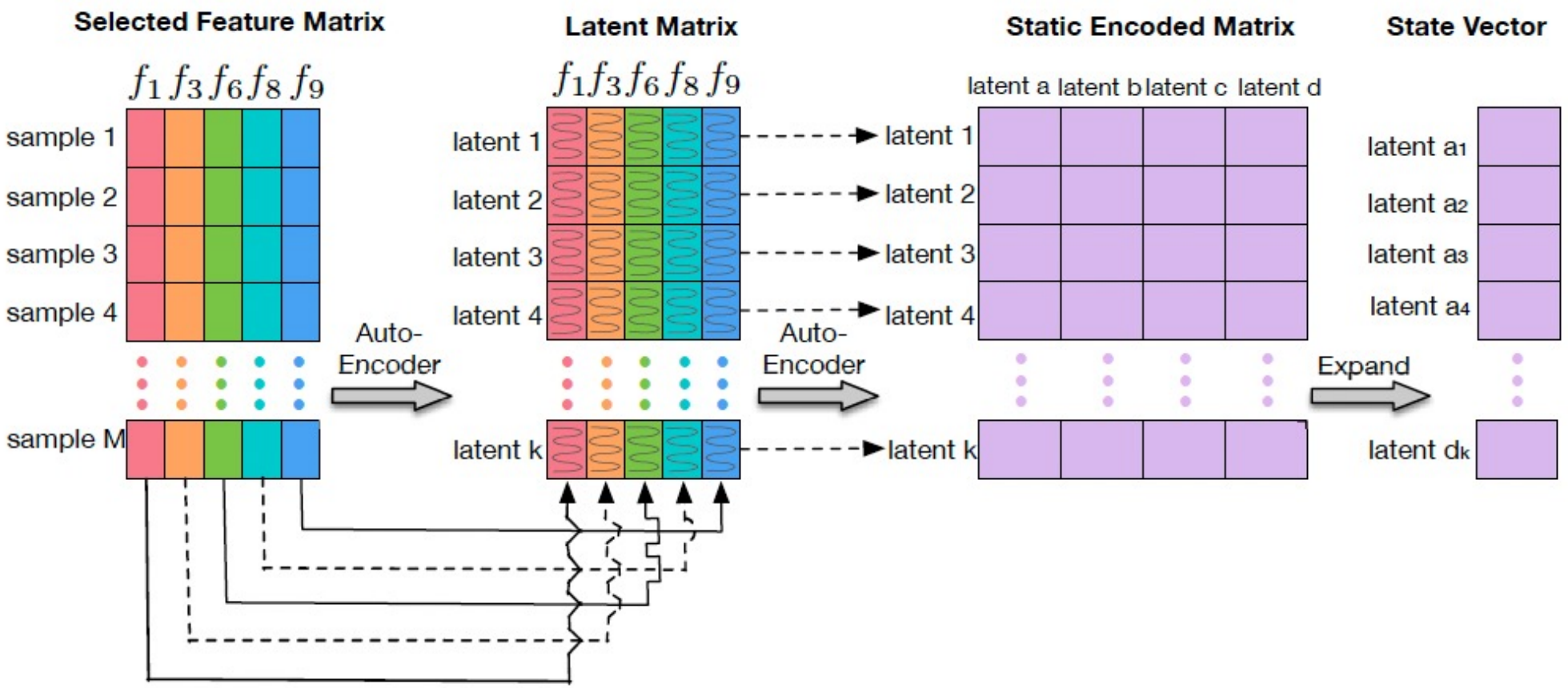
MARLFS

- State:
  - Meta descriptive statistics



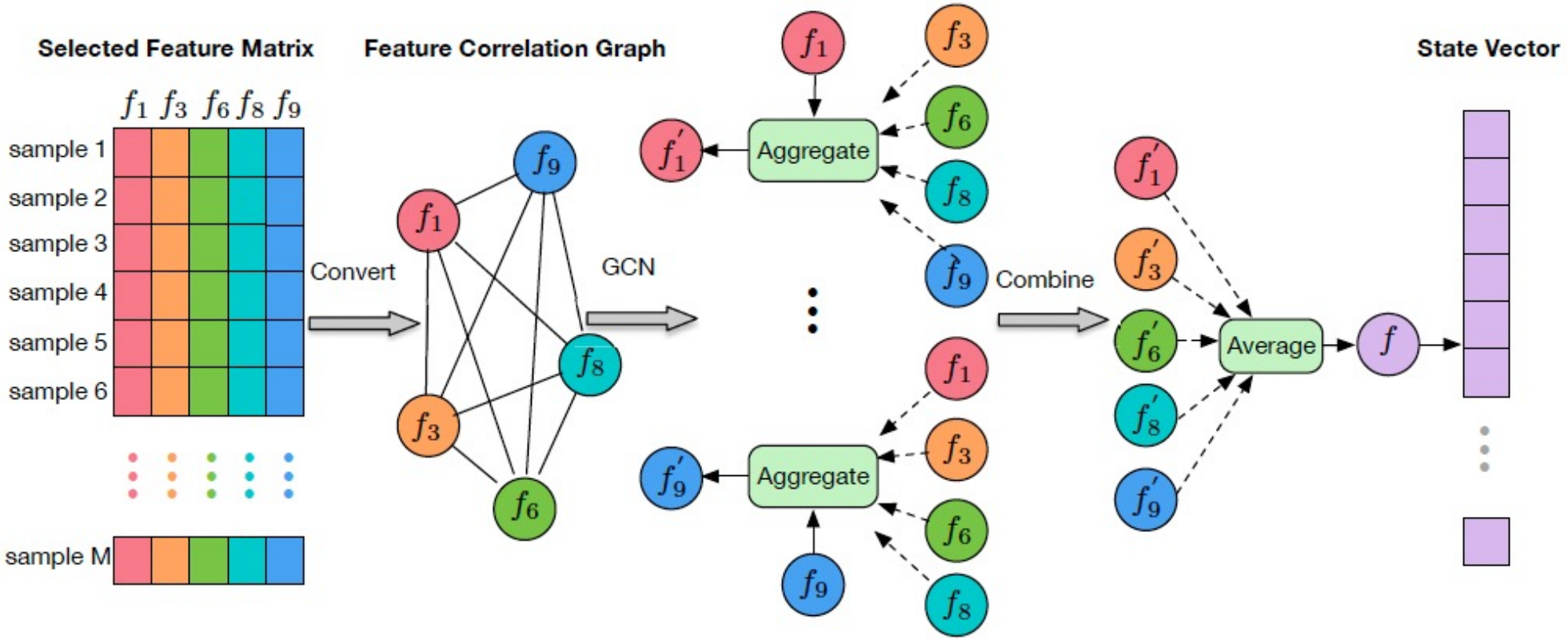
MARLFS

- State:
  - Meta descriptive statistics
  - Autoencoder



MARLFS

- State:
  - Meta descriptive statistics
  - Autoencoder
  - Graph convolutional network (GCN)



MARLFS

- State:
  - Meta descriptive statistics
  - Autoencoder
  - GCN

- Reward:
  - DRS accuracy
  - Information redundancy (x_i, x_j: selected features)
  - Information relevance (c: ground-truth)

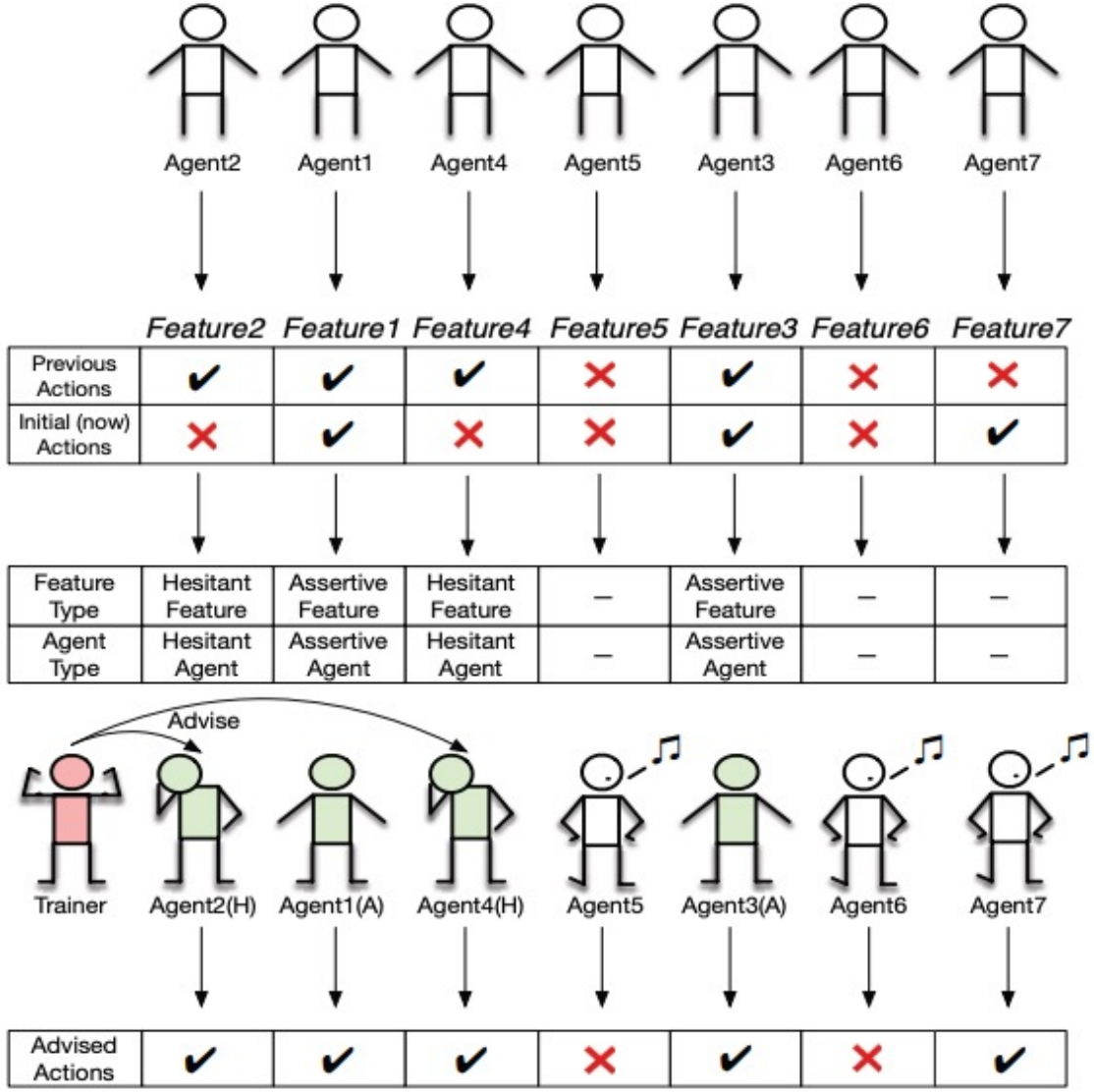
$$Rd = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i; x_j)$$

$$Rv = \frac{1}{|S|} \sum_{x_i \in S} I(x_i; c)$$

- Optimization: DQN

Subsequent works of MARLFS

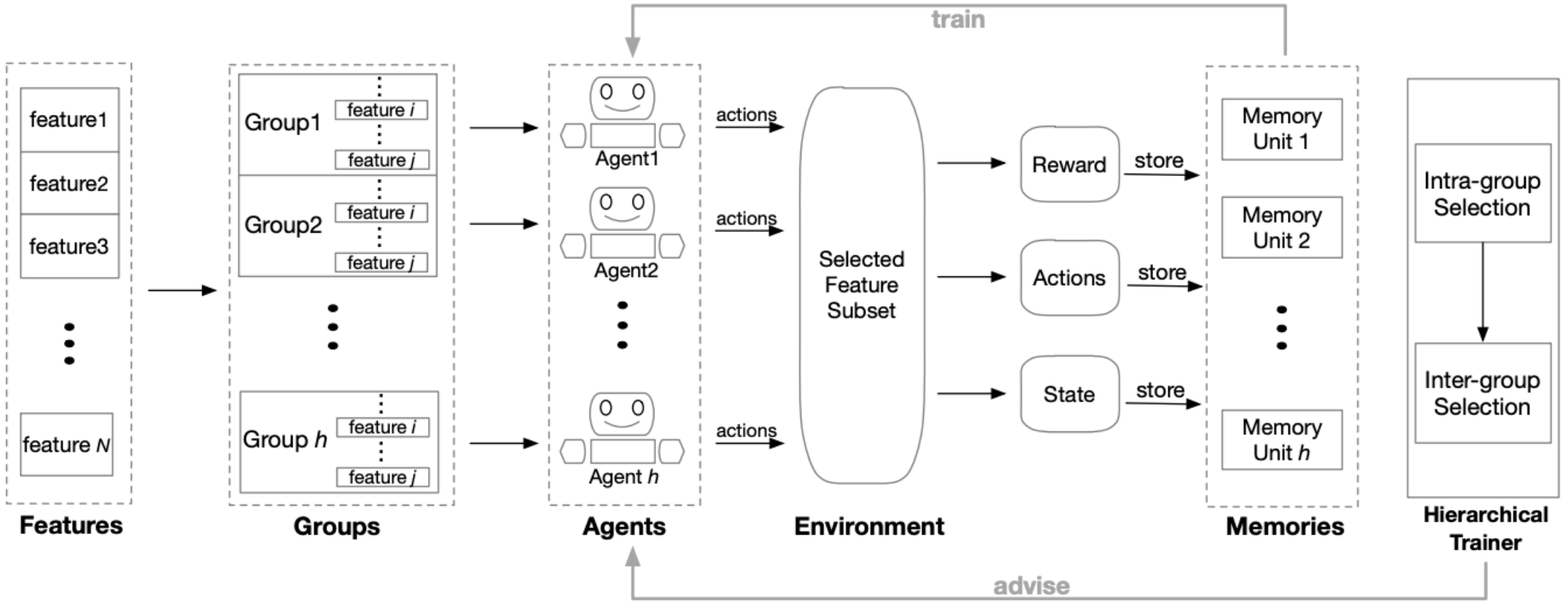
- AutoFS: Introduce external knowledge



AutoFS: Automated Feature Selection via Diversity-aware Interactive Reinforcement Learning, ICDM 2020

Subsequent works of MARLFS

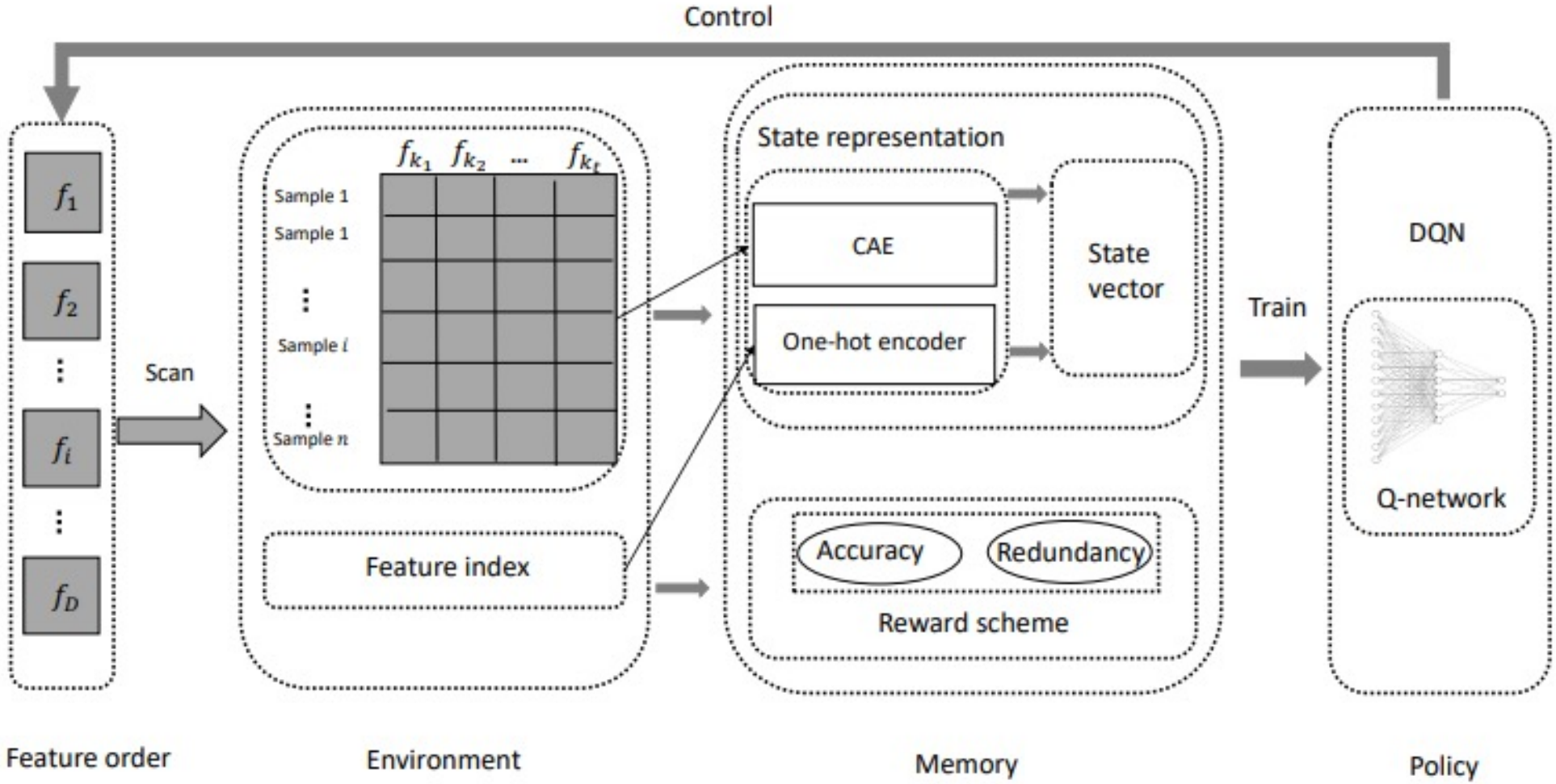
- AutoFS
- AutoGFS: Grouping feature fields



Group-based Feature Selection via Interactive Reinforcement Learning, SDM 2021

Subsequent works of MARLFS

- AutoFS
- AutoGFS
- SADRLFS: Multiple -> Single agent

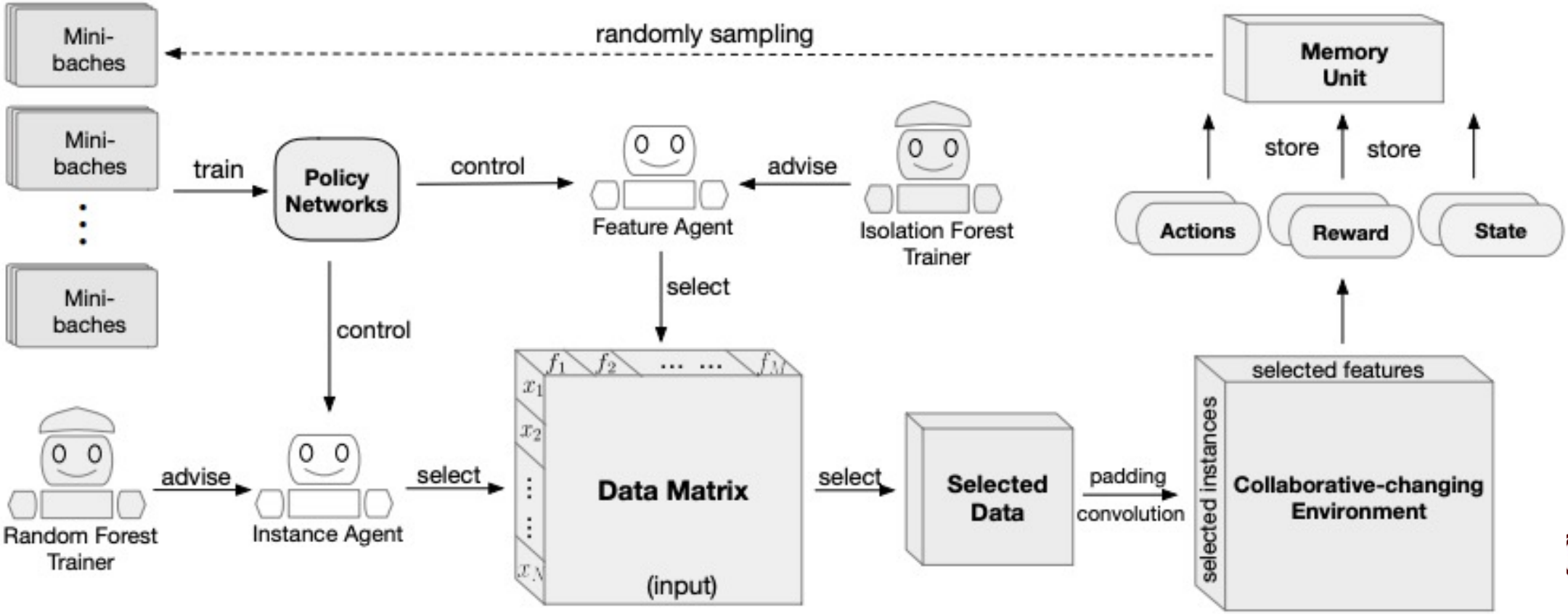


Simplifying Reinforced Feature Selection via Restructured Choice Strategy of Single Agent, ICDM 2020

Subsequent works of MARLFS

- AutoFS
- AutoGFS
- SADRLFS
- DAIRS: Simultaneously select features and samples

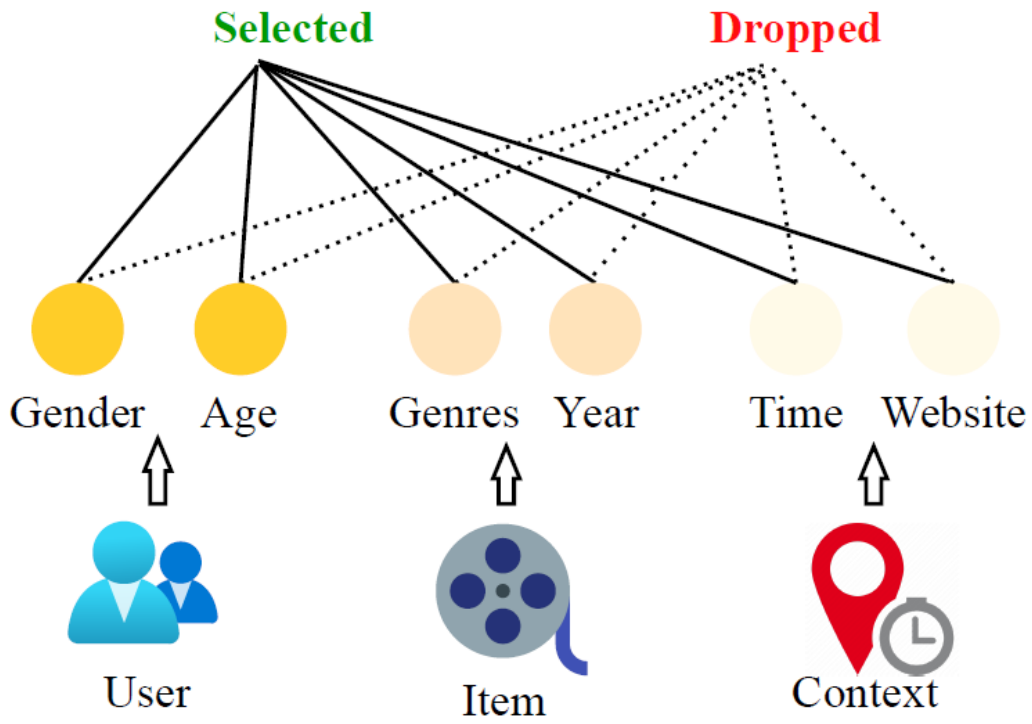
Feature and Instance Joint Selection: A Reinforcement Learning Perspective, SIGIR 2022



AutoField

- Motivation
  - Feeding all possible features → Extra embedding parameters
  - Manually selecting feature fields → Expert knowledge & Labor effort

- Target:
  - Automatically select feature fields
  - Field-level selection
  - Embedded

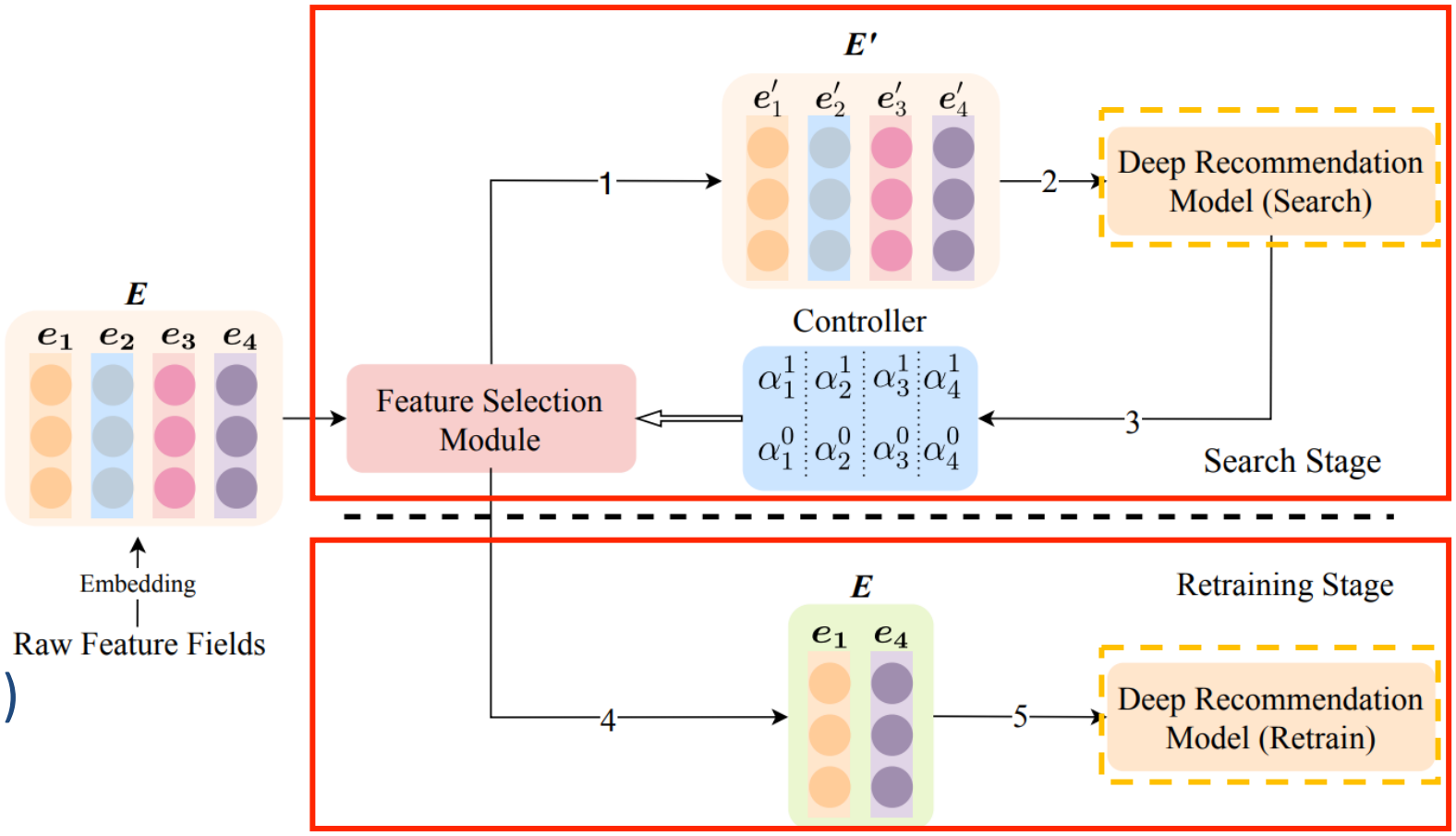


AutoField

- Data preparation: field embeddings

- Search stage:
  - Step 1: soft selection
  - Step 2: updating DRS (Search)
  - Step 3: updating controller

- Retraining stage:
  - Step 4: hard selection
  - Step 5: optimizing DRS (Retrain)



AutoField

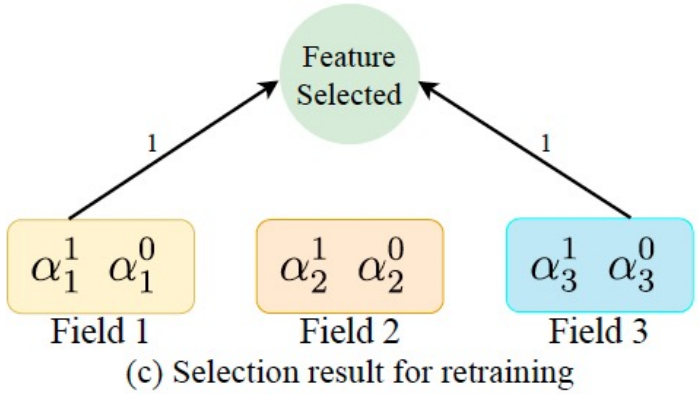
- Controller structure:
  - N parallel nodes for N feature fields
  - Node i contain two values: $\{\alpha_i^0, \alpha_i^1\}$

- In training:
  - α_i^1 of predictive feature fields would increase
  - α_i^0 of non-predictive feature fields would increase

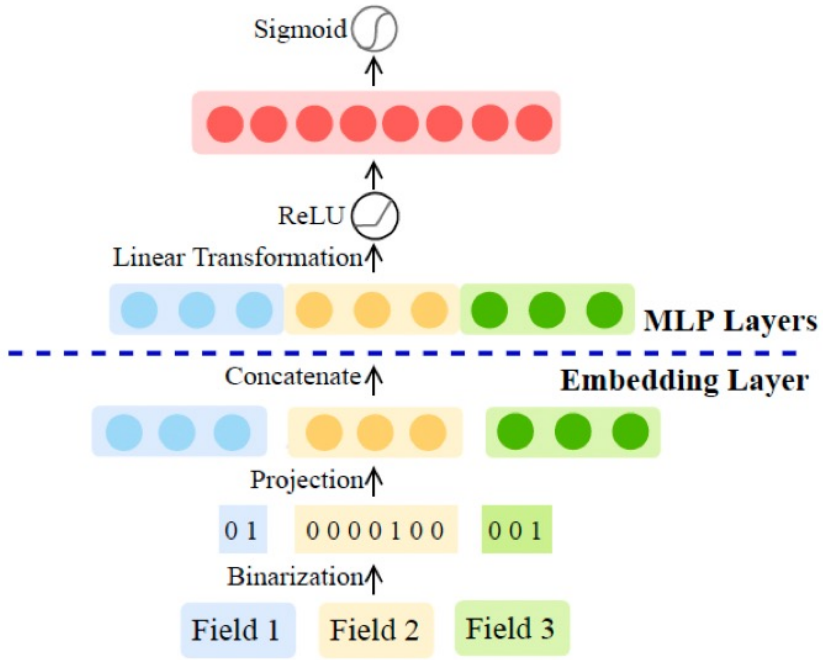
- $\{\alpha_i^0, \alpha_i^1\}$ is computed by Gumbel-Softmax
- Optimization: Gradient

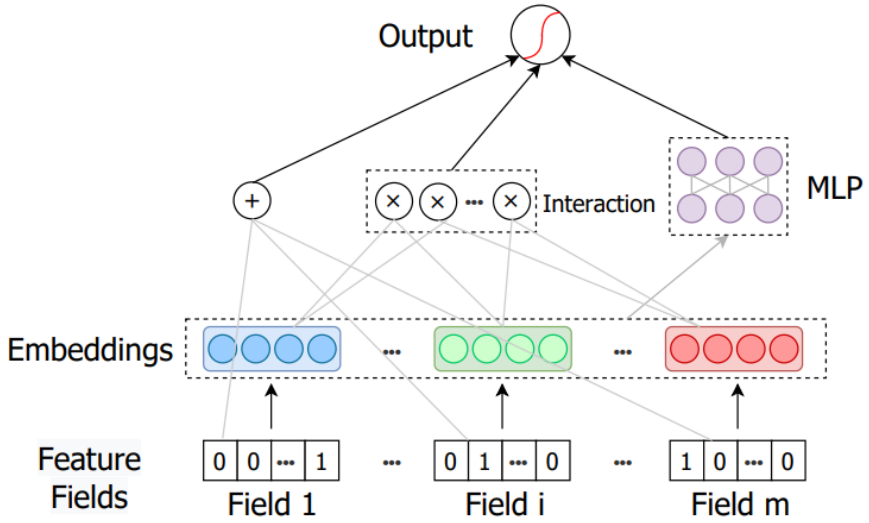
AutoField

- Retraining Stage
  - Only using K fields with highest score
  - Adapting model structures
  - Change subsequent DRS



(c) Selection result for retraining

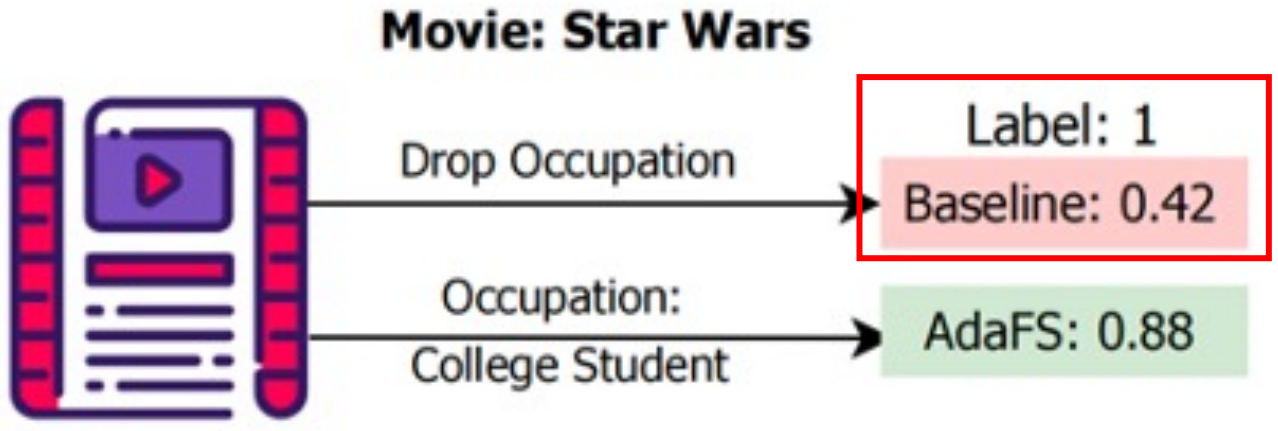




AdaFS

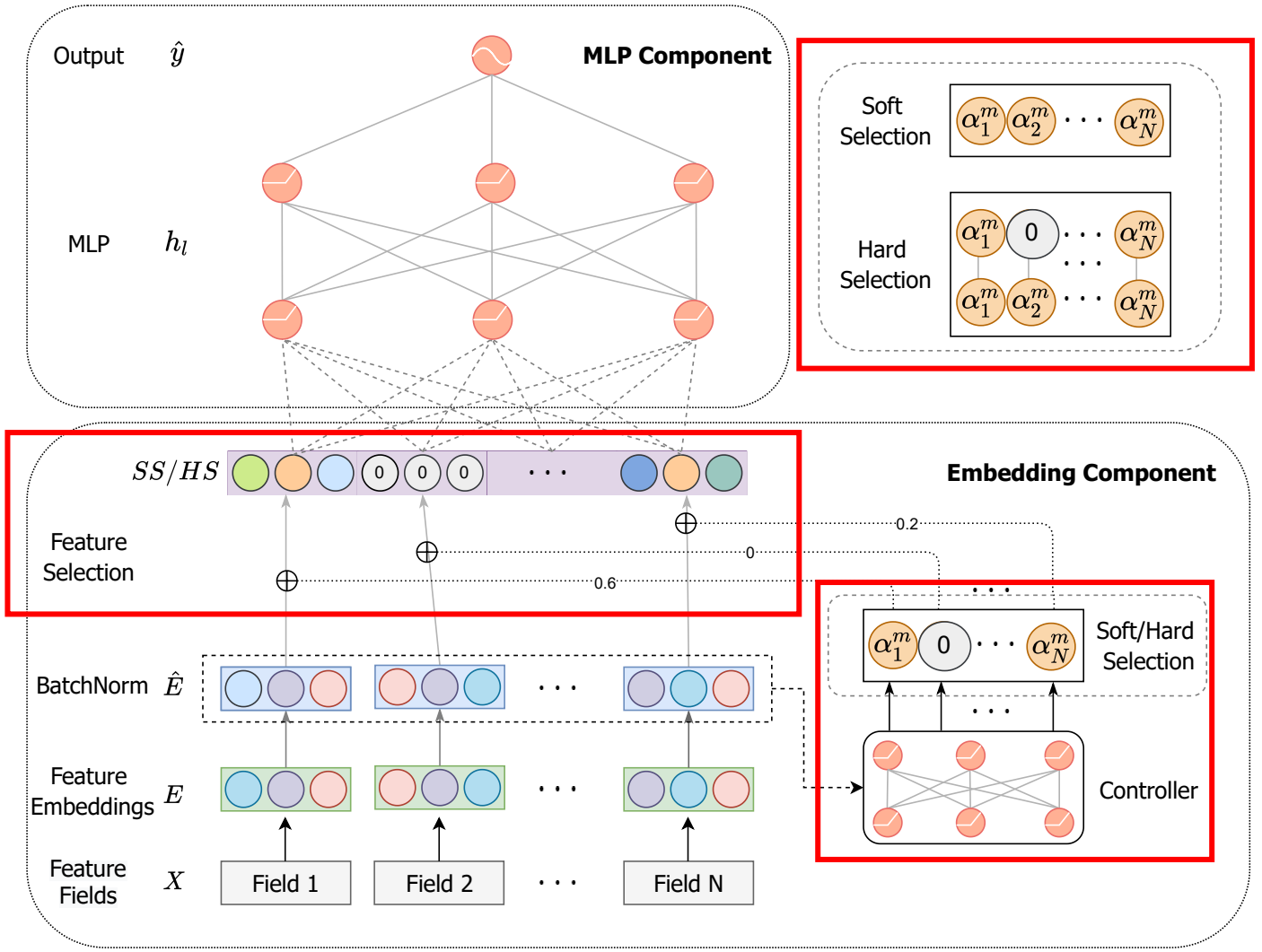
- Motivation:
  - Existing feature selection methods selects a fixed subset of features
  - Different user-item interactions → Different contributions

- Target:
  - Adaptively selecting the most predictive features
  - Field-level selection
  - Embedded



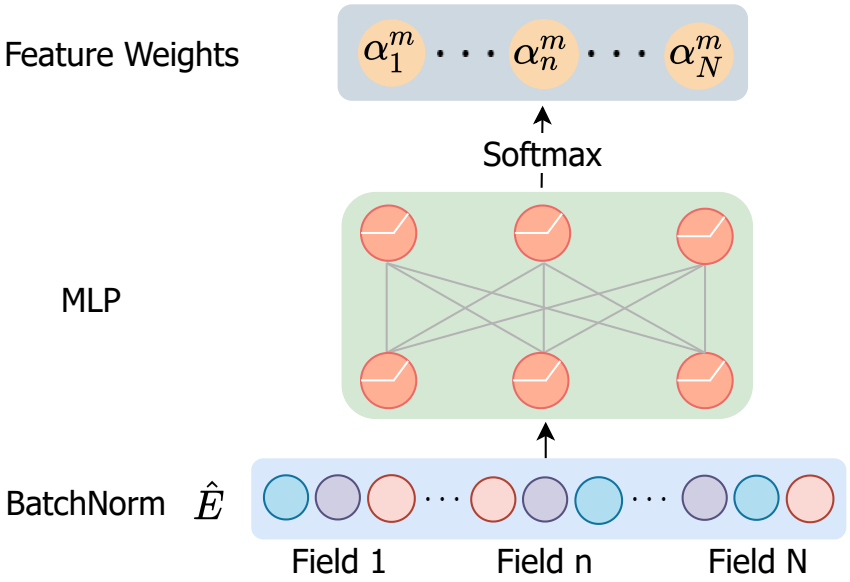
AdaFS

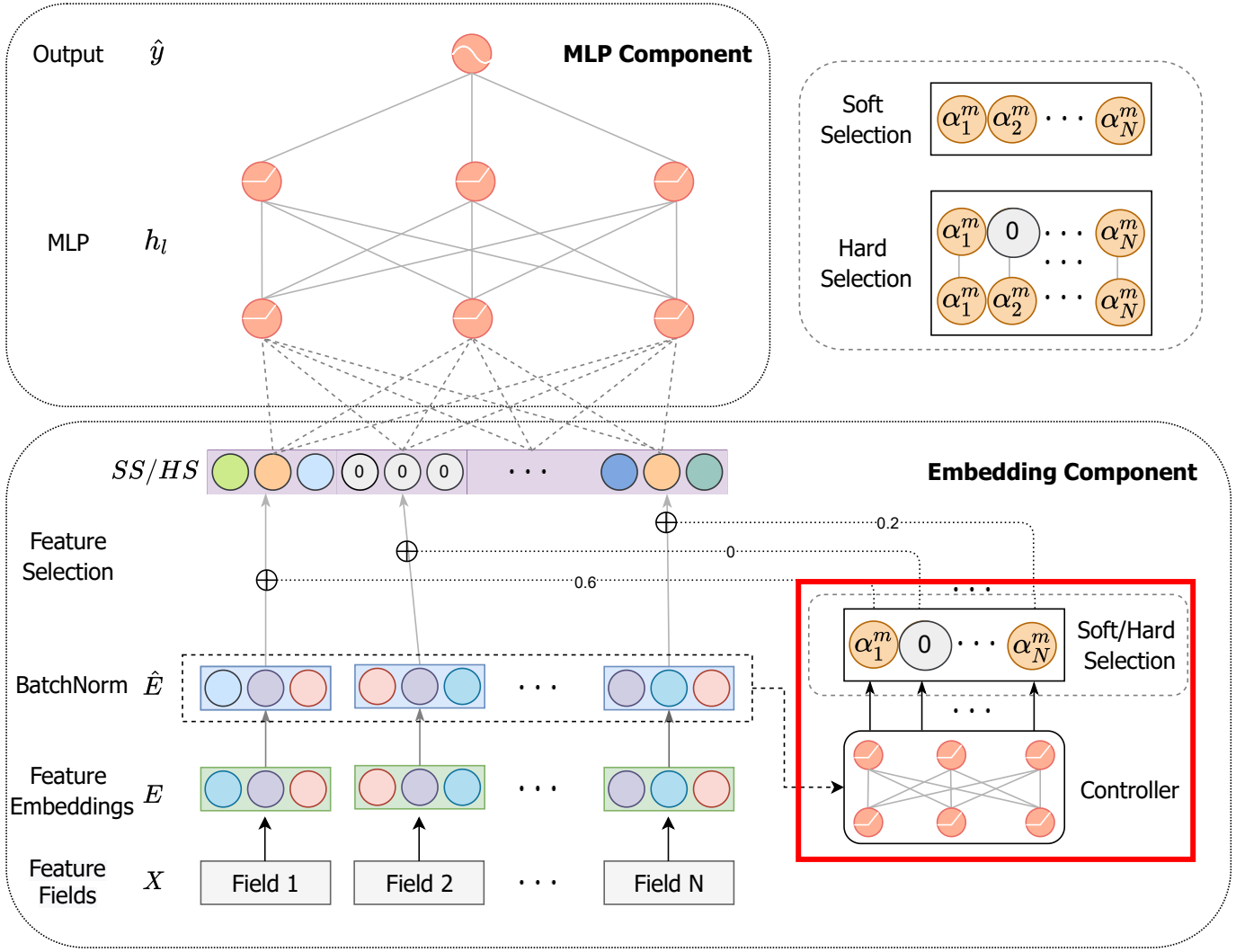
- DRS
  - Embedding component
  - Inference model (MLP)
- Controller
  - Adaptively scoring feature fields
- Feature selection
  - Hard selection
  - Soft selection



AdaFS

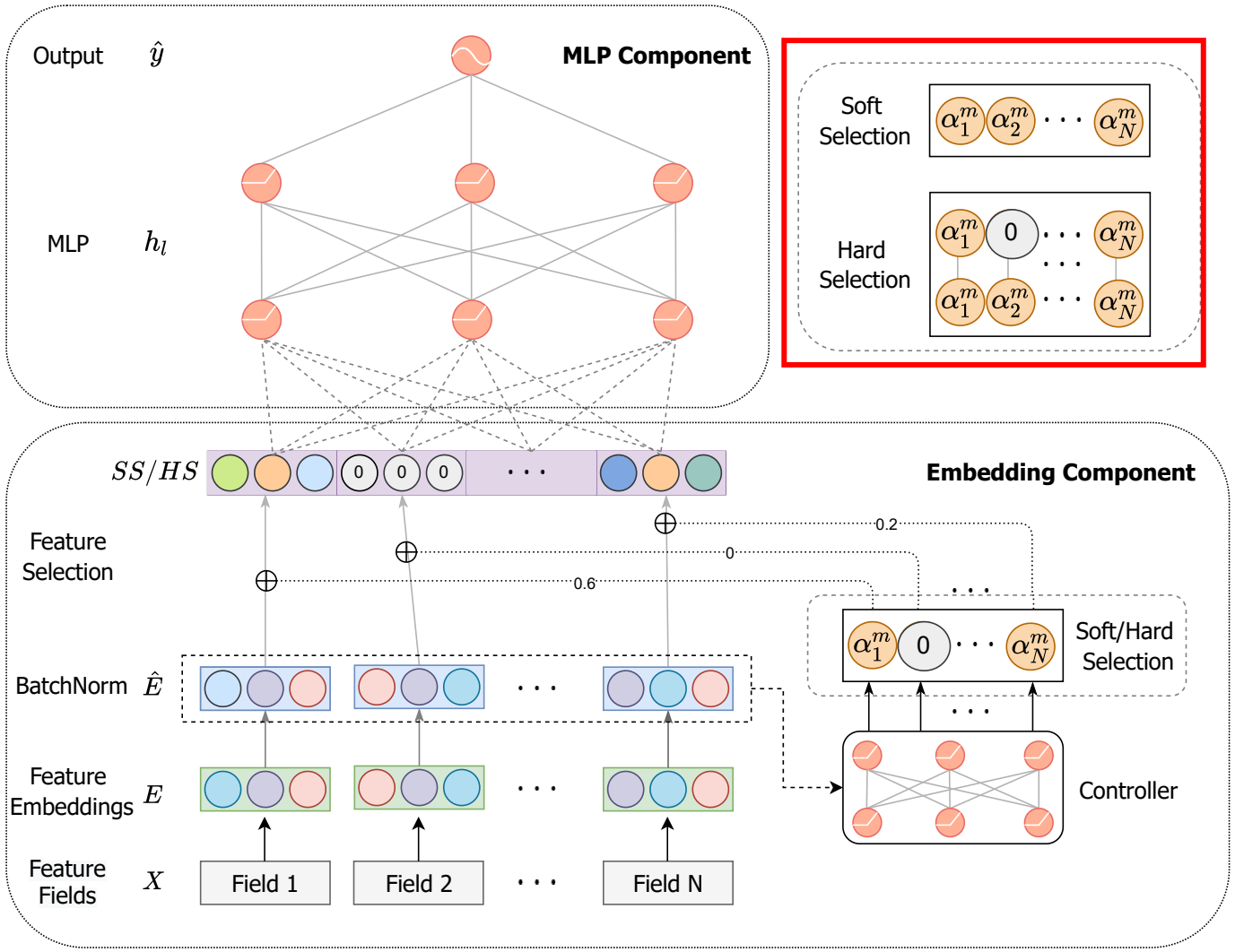
- Controller Network
  - Scoring the feature fields with weights
  - MLP + Softmax
  - BatchNorm:
    - Making embeddings comparable





AdaFS

- Soft Selection
  - Keeping all feature fields
  - Feature weights → Feature embeddings
- Hard Selection
  - Selecting fields with highest weights
- Optimization: Gradient
- Drawback:
  - Keeping all parameters



LPFS

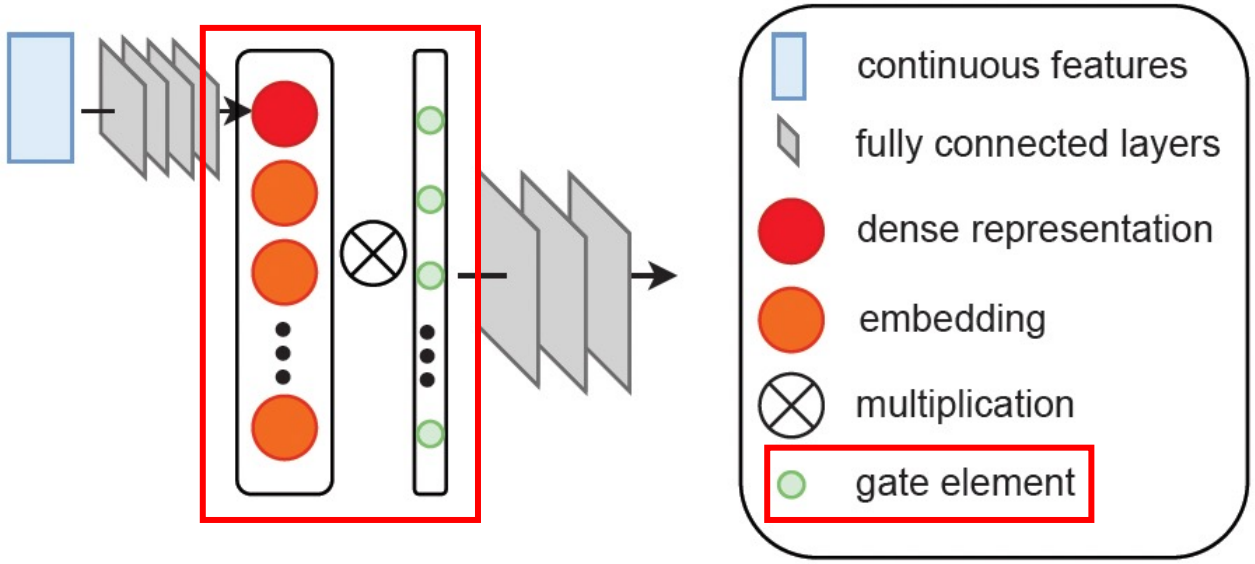
- Motivation
  - Removing feature fields with **non-zero** weights hurt the performance
  - Small weights do not guarantee redundant feature fields

- Target
  - Generate **exact-zero** weights for redundant feature fields
  - Field-level selection
  - Embedded

LPFS

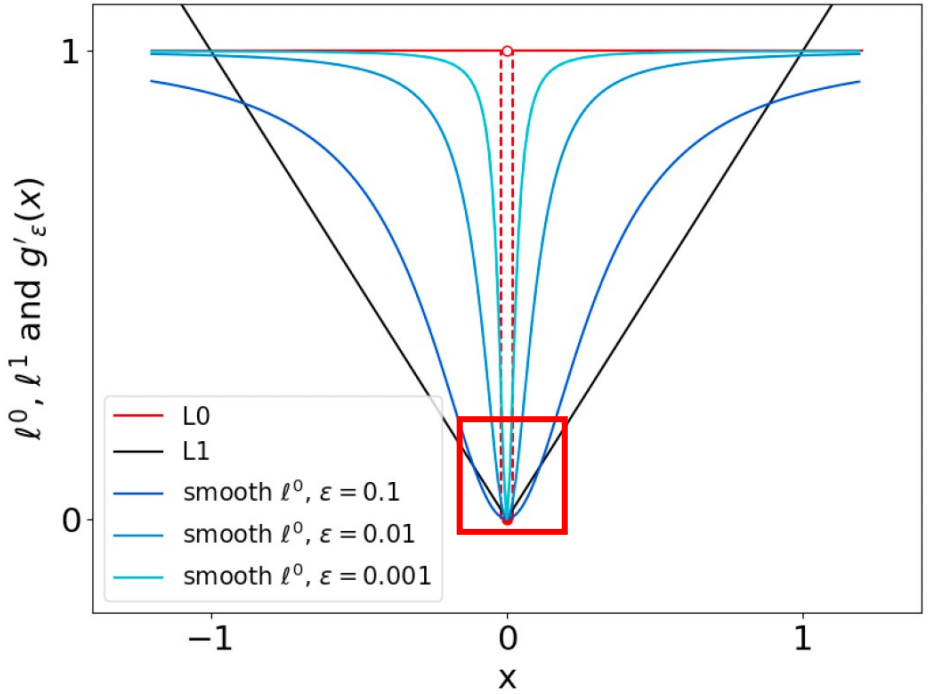
- Problem formulation:

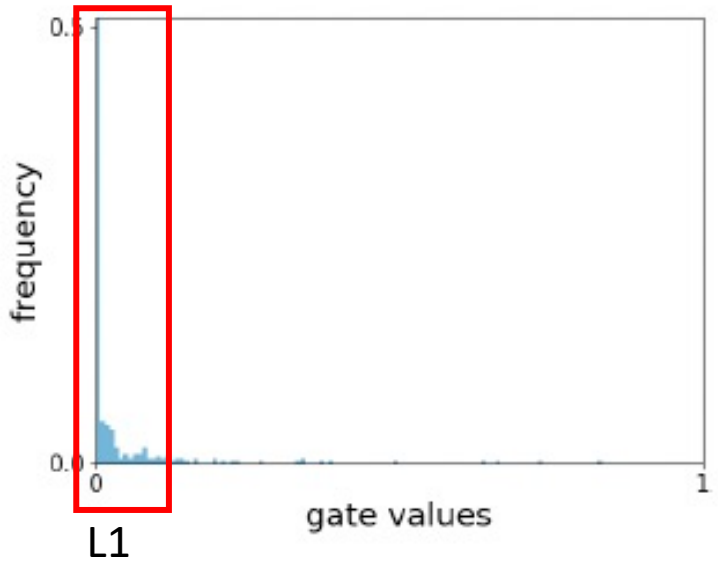
$$y = f(w; e) \longrightarrow \begin{array}{l} g(x) = [g(x_1), g(x_2), ..., g(x_N)] \\ y = f(w; g(x)e) \end{array}$$



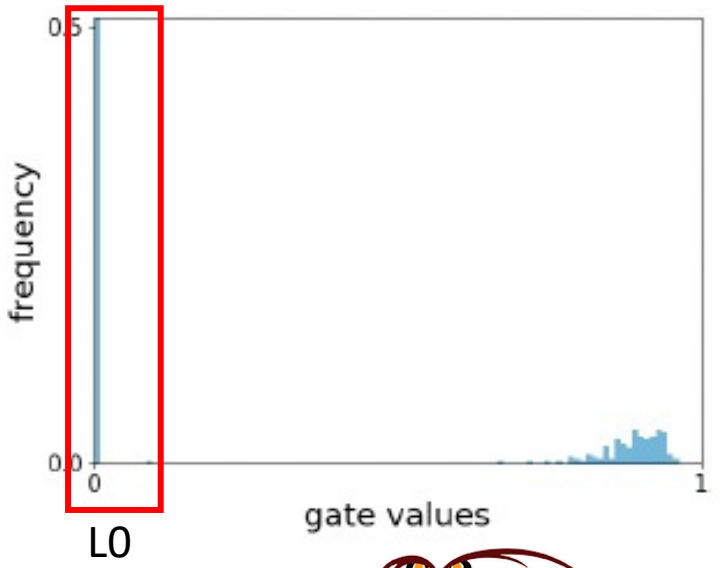
- LPFS:

$$g_\epsilon(x) = \frac{x^2}{x^2 + \epsilon}$$





L1

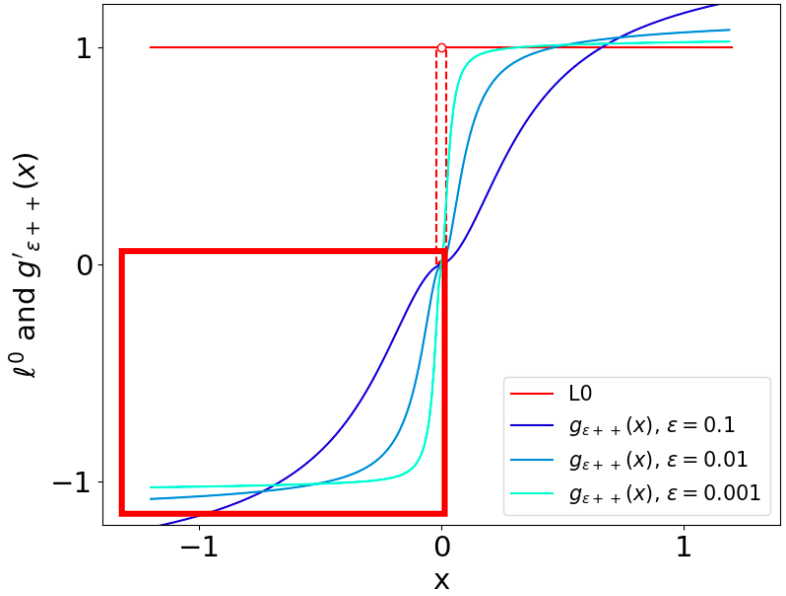


L0

LPFS

- Problem of LPFS: Gradient at $x = 0$ is 0.
  - Sensitive to noise
  - Evolving user behavior in DRS

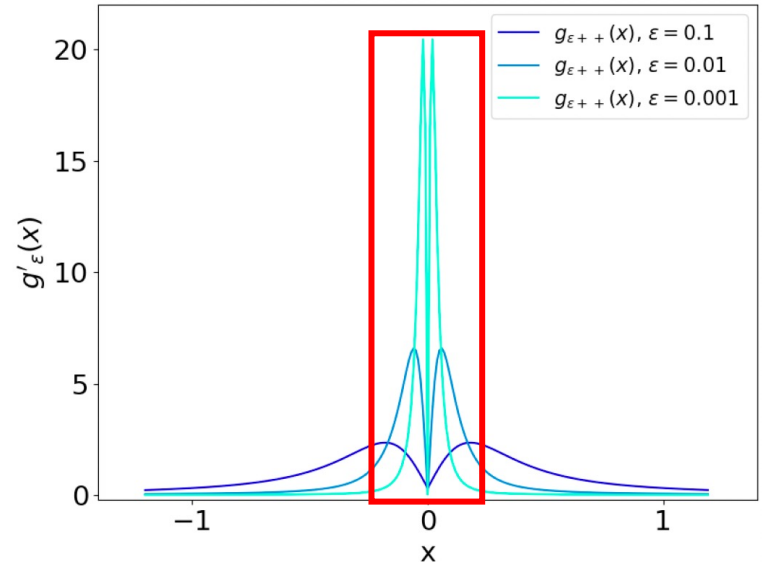
$$g'_\epsilon(x) = \frac{2x\epsilon}{(x^2+\epsilon)^2}$$



- LPFS++:

$$g_{\epsilon++}(x) = \begin{cases} \frac{x^2}{x^2+\epsilon} + \alpha\epsilon^{1/\tau}\arctan(x), & x \geq 0 \\ -\frac{x^2}{x^2+\epsilon} + \alpha\epsilon^{1/\tau}\arctan(x), & x < 0 \end{cases}$$

$$g'_{\epsilon++}(x) = \frac{2|x|\epsilon}{(x^2+\epsilon)^2} + \frac{\alpha\epsilon^{1/\tau}}{x^2+1}$$

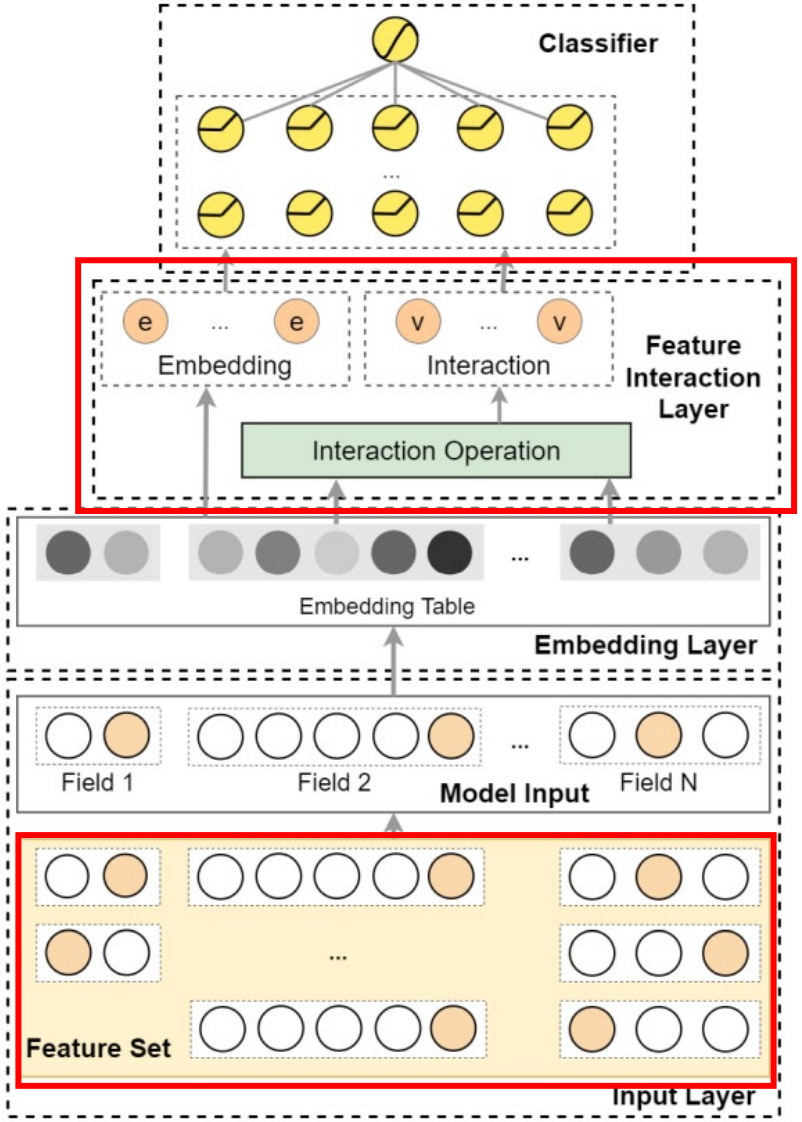


- Optimization: Gradient

OptFS

- Motivation
  - Field-level selection is too coarse
    - E.g., “user ID”
  - Feature interaction considers redundant features

- Target
  - Conduct **feature-level** selection
  - Simultaneously finish **feature interaction search**
  - Embedded



OptFS

- Model input

$$\mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n] = [\mathbf{x}_{k_1}, \mathbf{x}_{k_2}, \cdots, \mathbf{x}_{k_n}], \ 1 \leq k_i \leq m,$$

- Feature selection:

$$\mathbf{e}^{\mathbf{g}}_{k_i} = \mathbf{g}_{k_i} \odot \mathbf{e}_{k_i} = \mathbf{g}_{k_i} \odot (\mathbf{E} \times \mathbf{x}_{k_i}).$$

- Search space: Gates $g_{k_i} \in \{0,1\}$

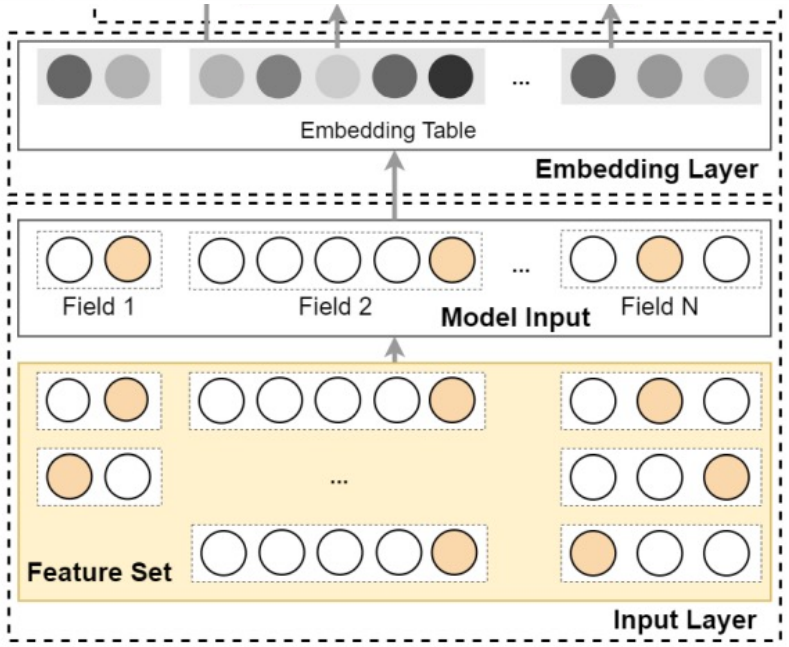
- Feature interaction selection:

$$\mathbf{g}'_{(k_i,k_j)} = \mathbf{g}_{k_i} \times \mathbf{g}_{k_j}, \quad \mathbf{v}_{(i,j)} = \mathcal{O}(\mathbf{e}_i, \mathbf{e}_j)$$

- Prediction:

$$\hat{y} = \mathcal{H}((\mathbf{g} \times \mathbf{g} \odot \mathbf{v}) \oplus \mathcal{G}(\mathbf{g} \odot \mathbf{e})),$$

Model	$\mathcal{G}(\cdot)$	$\mathcal{O}(\cdot)$	$\mathcal{H}(\cdot)$
FM [26]	null	inner product	null
DeepFM [7]	MLP	inner product	average
DCN [31]	MLP	cross network	average
IPNN [24]	null	inner product	MLP
OPNN [24]	null	outer product	MLP
PIN [25]	null	MLP	MLP



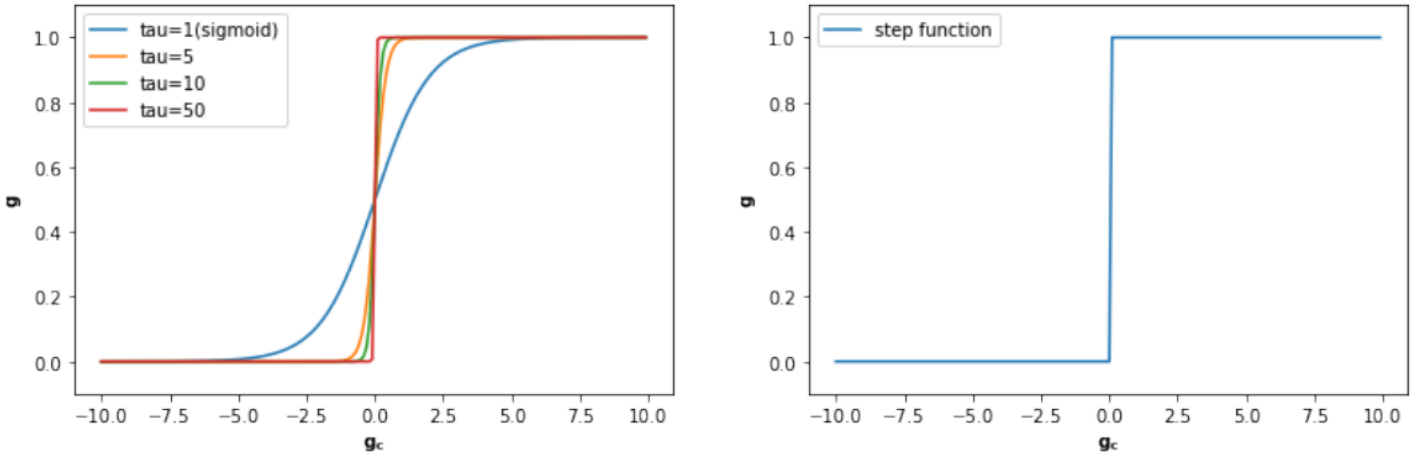
OptFS

- Searching

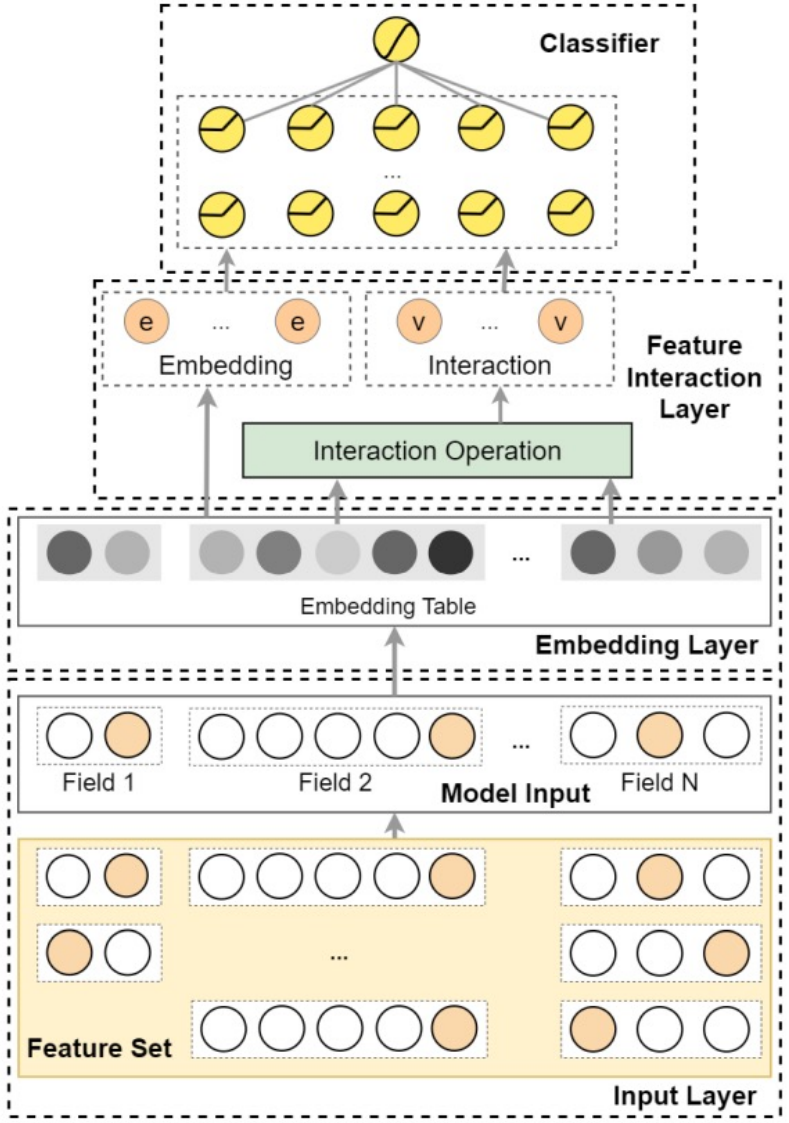
$$\mathbf{g} = \frac{\sigma(\mathbf{g}_c \times \tau)}{\sigma(\mathbf{g}_c^{(0)})}, \ \tau = \gamma^{t/T}$$

- Retraining

$$\mathbf{g} = \begin{cases} 0, & \mathbf{g}_c \leq 0 \\ 1, & \text{otherwise} \end{cases}$$



(a) Searching Stage (b) Re-training Stage



- Optimization: Gradient

Selection of raw features

Model	Granularity	Gating/Scoring	Search Strategy
FSTD	Field-level	Temporal Difference	RL (TD + UCB)
MARLFS	Field-level	None	RL (DQN)
AutoField	Field-level	Continuous	Gradient
AdaFS	Field-level	Continuous	Gradient
LPFS	Field-level	Zero/Non-zero	Gradient (L0)
OptFS	Feature-level	Approx. zero	Gradient (L0)

- Reinforcement learning methods consider the problem of feature selection as a Markov decision process. They are less prone to overfitting since they usually overall performance of the model when designing reward functions;

- Gradient-based approaches are more practical to real-world recommender systems owing to their efficiency and simplicity. In addition, they are flexible to be applied to various recommendation models and datasets.

Table of Contents

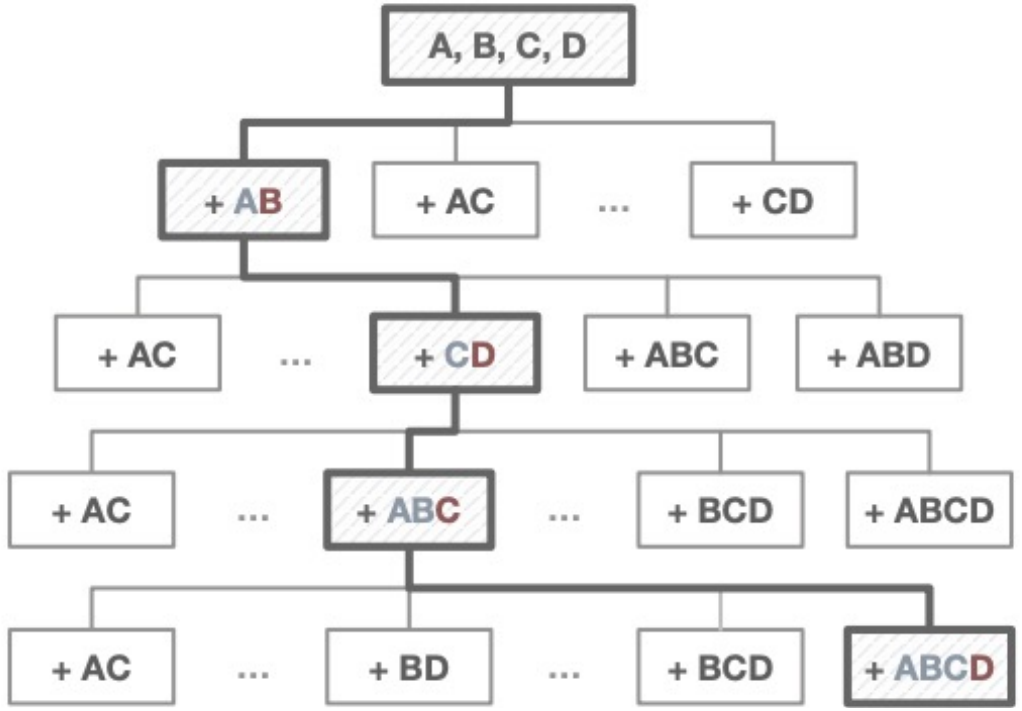
- DRS Feature Selection
  - Selection from raw features
  - Selection from generated features

AutoCross

- Target:
  - Select crossed features
  - Field-level
  - Wrapper

- Feature set generation:

  {A, B, C, D}
  → {A, B, C, D, AB}
  →{A, B, C, D, AB, CD}
  →{A, B, C, D, AB, CD, ABC}
  →{A, B, C, D, AB, CD, ABC, ABCD}



- Feature evaluation:
  - Field-wise logistic regression

$$P(y = 1|\mathbf{x}) = Sigmoid(\boldsymbol{w}_{new}\mathbf{x}_{new} + \boldsymbol{w}_c\mathbf{x}_c)$$

- Optimization: Beam search

GLIDER

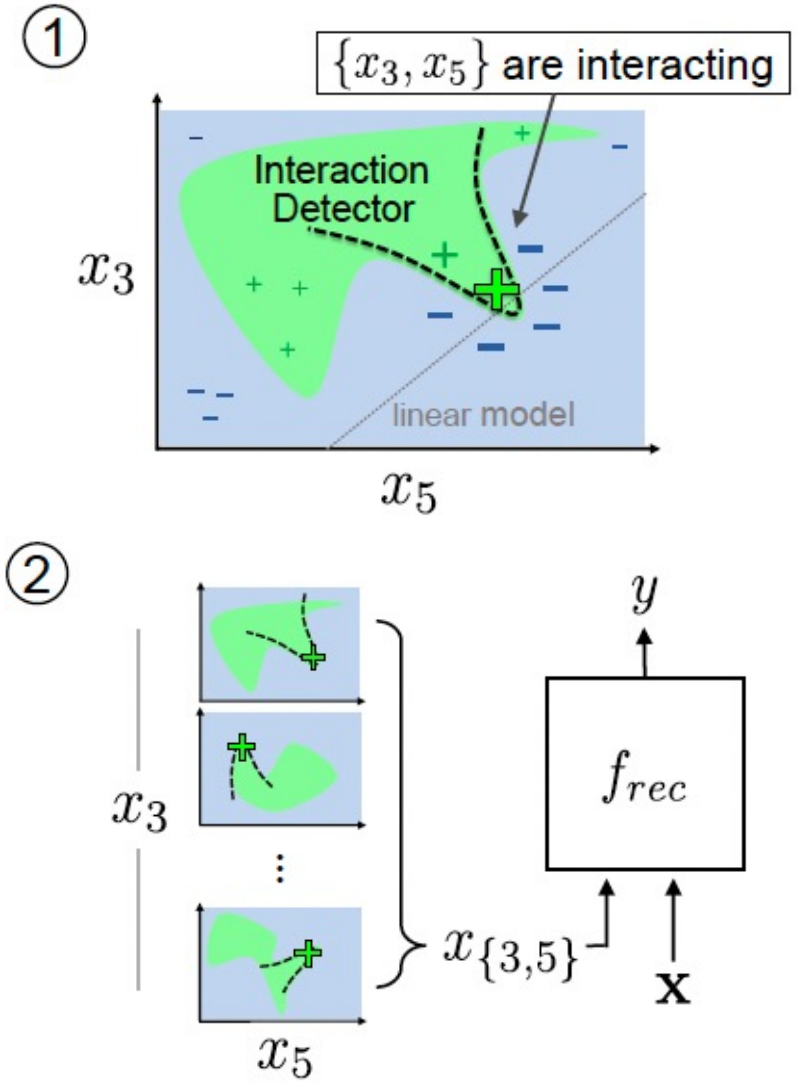
- Problem:
  - Select generated features for black-box DRS
  - Field-level
  - Filter
- Step 1: Global interaction detection
  - LIME: Perturb the input feature

$$\mathcal{D} = \{(\tilde{\mathbf{x}}_i, y_i) \mid y_i = f(\xi(\tilde{\mathbf{x}}_i)), \tilde{\mathbf{x}}_i \in \{0,1\}^d\}.$$

  - Gradient NID: Detect the interaction

$$\omega(\mathcal{I}) = \left(\frac{\partial^{|\mathcal{I}|} g(\tilde{\mathbf{x}})}{\partial \tilde{x}_{i_1} \partial \tilde{x}_{i_2} \dots \partial \tilde{x}_{i_{|\mathcal{I}|}}} \right)^2,$$

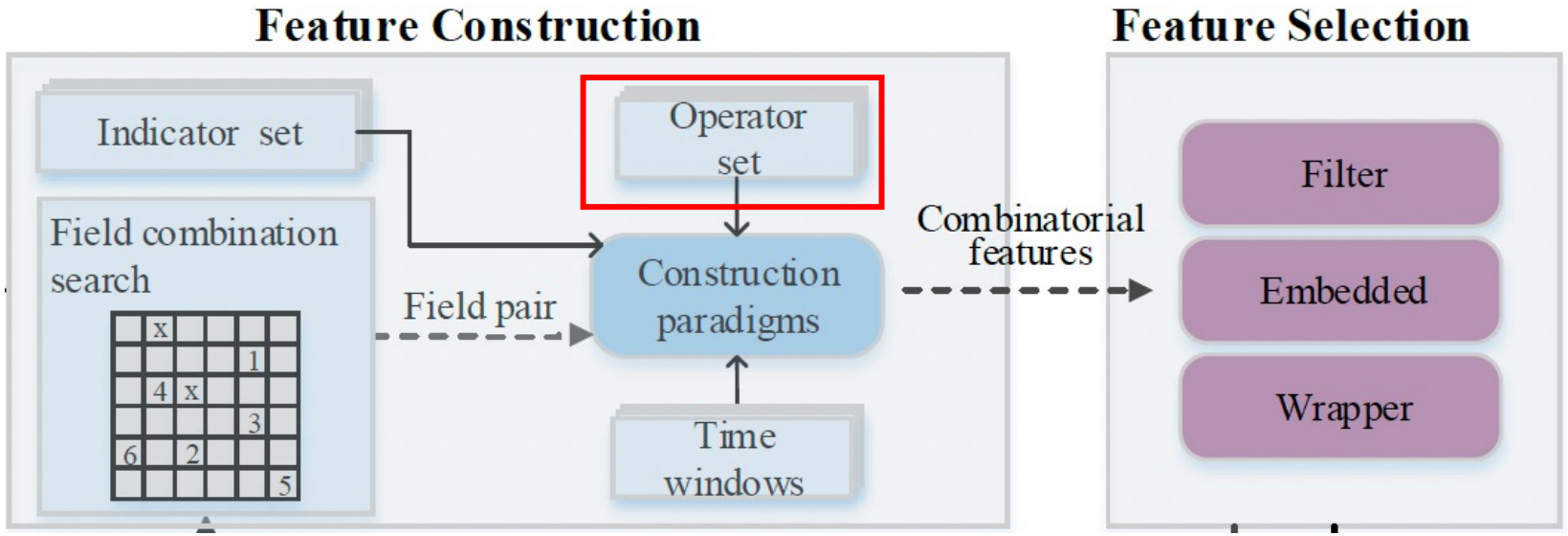
- Step2: Construct DRS and retrain
  - Same model: enhancement
  - Different models: Teacher-student
- Optimization: Gradient



Feature Interaction Interpretability:A Case for Explaining Ad-recommendation Systems via Neural Interaction Detection, ICLR 2020

AEFE

- Target:
  - Construct and select combinatorial features
  - Field-level
  - Combination of Filter, Embedded, Wrapper
- Procedure:
  - 1-Filter: Drop features with low variance
  - 2-Embedded: Generate feature importance by GBDT/RF
  - 3-Wrapper: Add features in a cascaded manner, greedy search



AEFE: Automatic Embedded Feature Engineering for Categorical Features, 2021

Selection of generated features

Model	Combination	Granularity	Gating/Scoring	Search Strategy
AutoCross	Wrapper	Field-level	None	Beam Search
GLIDER	Filter	Field-level	NID	Gradient
AEFE	All	Field-level	Continuous	Greedy Search

- Selectively learning the generated features can bring great precision improvement to prediction. They are highly interpretable, which is helpful for digging deep into the underlying relationship of the data;
- Compared with selection from raw features, generated feature selection usually has a much larger search space. Researchers usually adopt greedy search methods, which suffer from the heavy storage pressure and time-consuming process. It is highly desirable for efficient AutoML techniques to facilitate the selection from generated features.

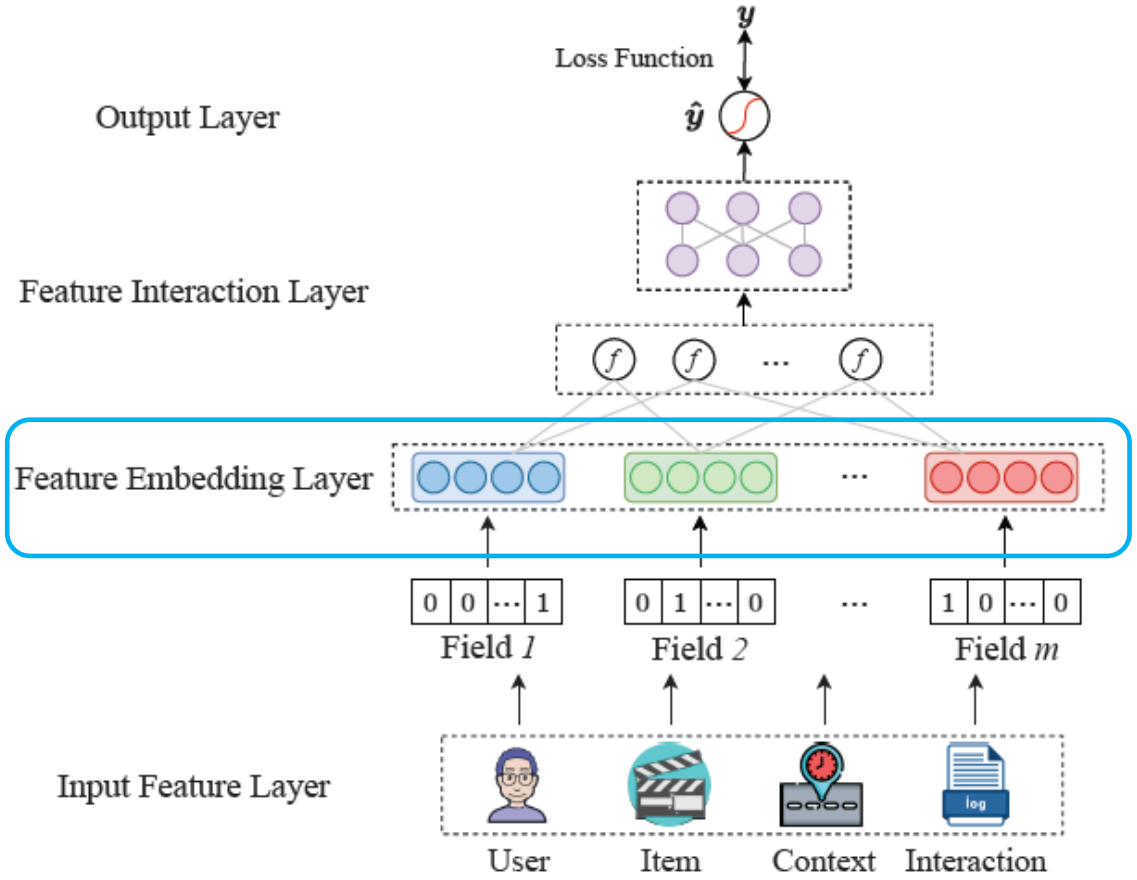
Summarize AutoML for FS

	Model	Combination	Granularity	Gating/Scoring	Search Strategy
Raw Feature	FSTD	Filter/Wrapper	Field-level	AUC	RL (UCB)
	MARLFS	Embedded	Field-level	None	RL (DQN)
	AutoField	Embedded	Field-level	Continuous	Gradient
	AdaFS	Embedded	Field-level	Continuous	Gradient
	LPFS	Embedded	Field-level	Zero/Non-zero	Gradient (L0)
	OptFS	Embedded	Feature-level	Approx. zero	Gradient (L0)
Generated Feature	AutoCross	Wrapper	Field-level	AUC	Beam Search
	GLIDER	Filter	Field-level	NID	Gradient
	AEFE	All	Field-level	Continuous	Greedy Search

Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
  - Full Embedding Search
  - Column-based Embedding Search
  - Row-based Embedding Search
  - Column&Row-based Embedding Search
  - Combination-based Embedding Search
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

Background



Embedding Table

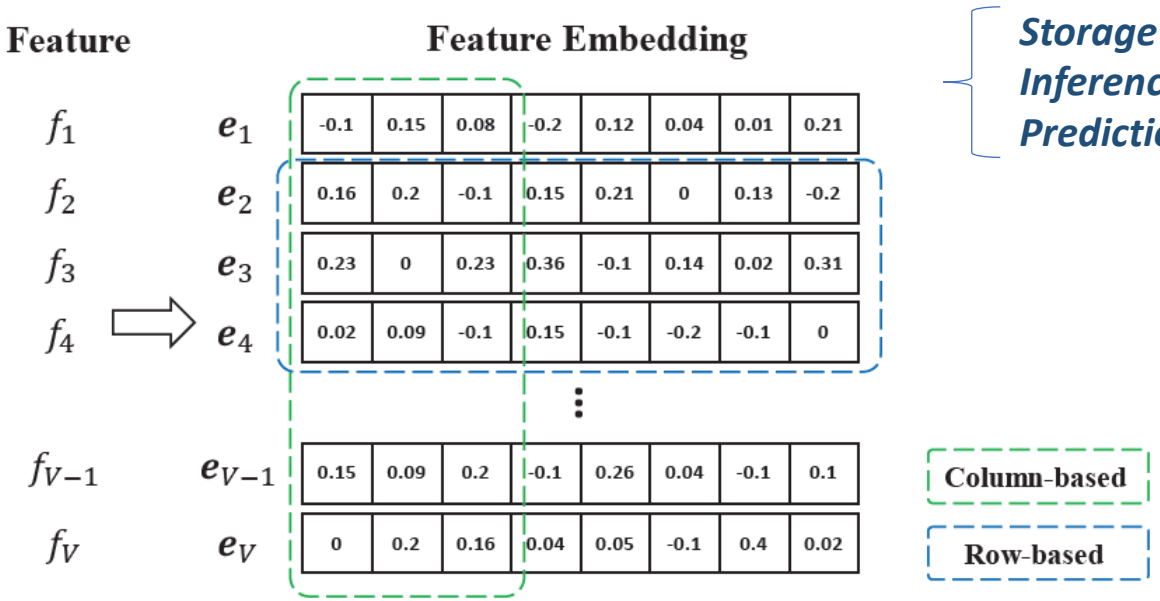
$$E \in R^{V \times d}$$

Number of Feature Values

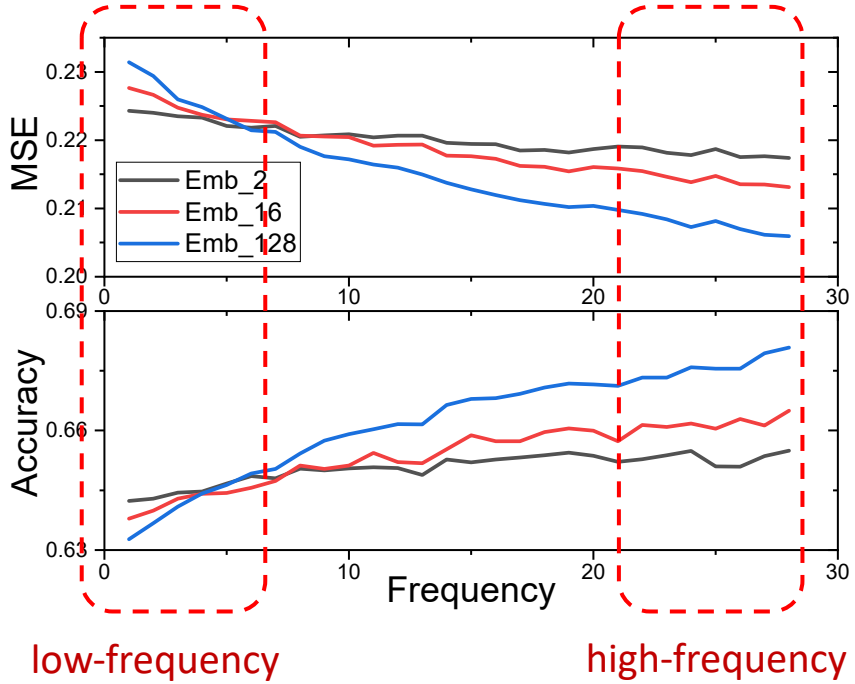
Embedding Size

- The embedding layer is used to map **the high-dimensional features** into a **low-dimensional latent space**.
- The **cornerstone of the DRS,** as the number of parameters in DRS is concentrated in the embedding table.

Background

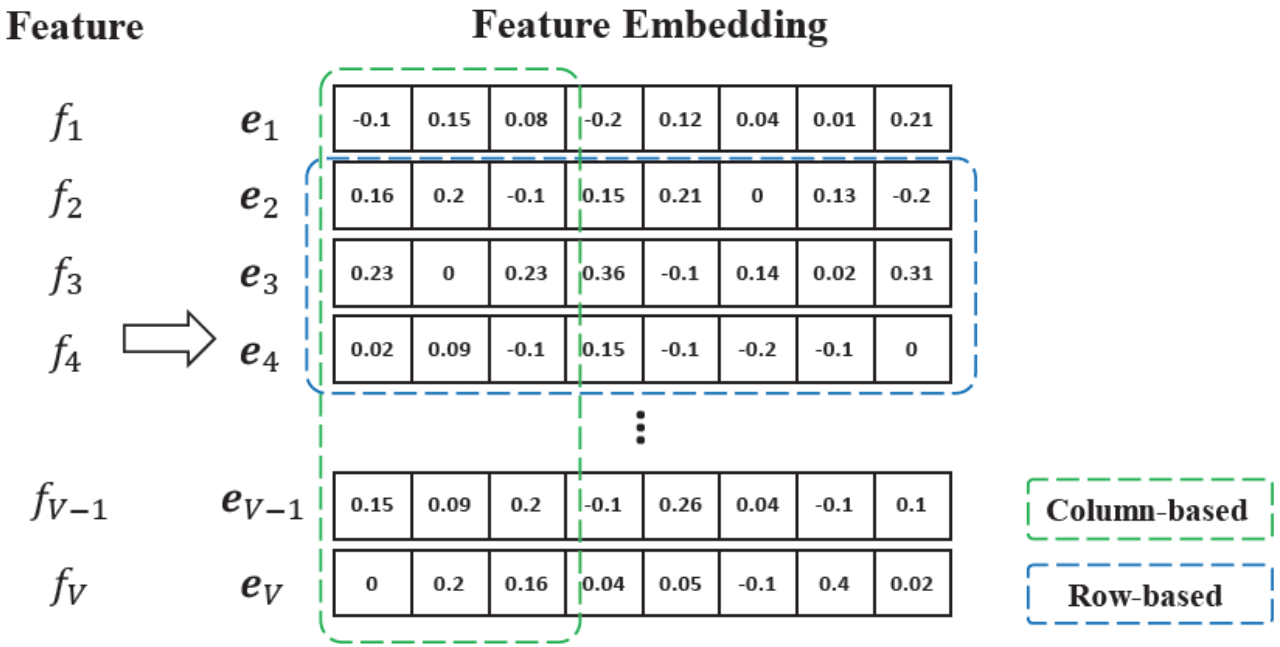


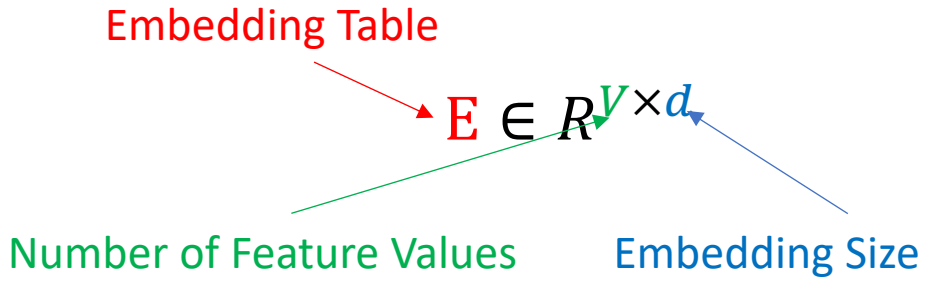
Storage capacity
Inference efficiency
Prediction accuracy



To **improve the prediction accuracy, save storage space and reduce model size**, AutoML-based solutions are proposed for the learning of feature embedding.

Background





- Full Embedding Search
- Column-based Embedding Search
- Row-based Embedding Search
- Column & Row-based Embedding Search
- Combination-based Embedding Search

Full Embedding Search

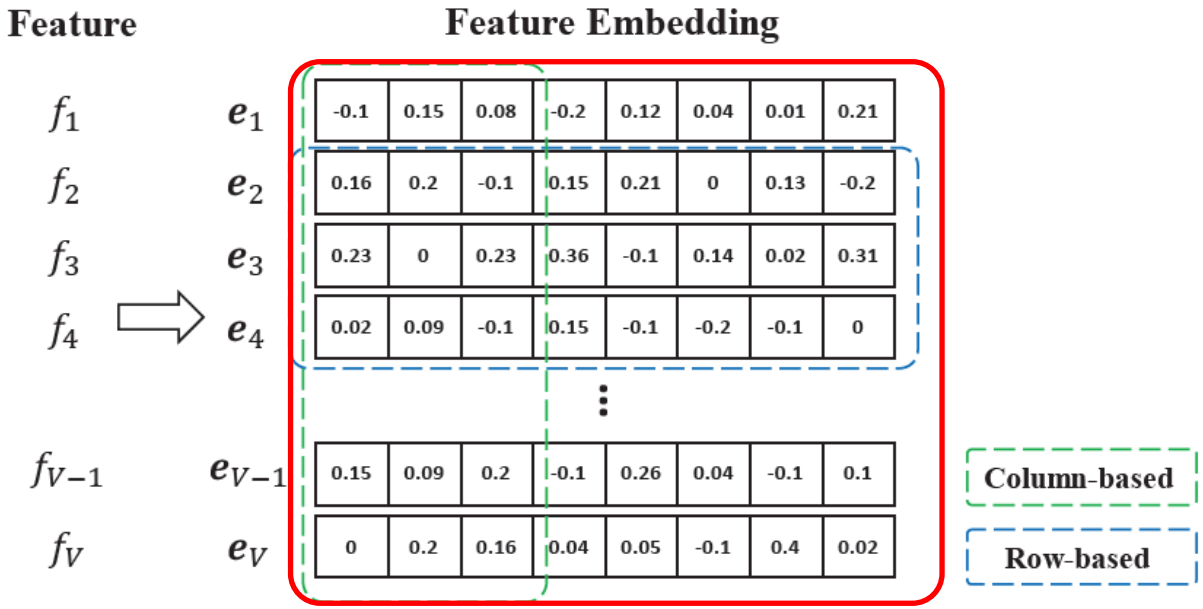
- The finest-grained embedding dimension search over the original embedding table
- Search the optimal embedding dimension for each feature value
  - ✓ Fully consider the impact of each feature embedding dimension on the prediction results
  - ✓ High-frequency or low-frequency feature values can be assigned with different dimensions
  - x Search space is huge
  - x Hard to reduce the storage space

Embedding Table

$$E \in R^{V \times d}$$

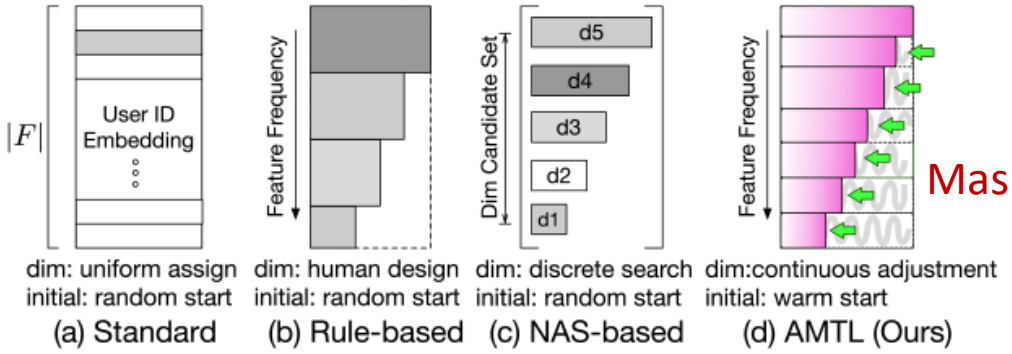
Number of Feature Values

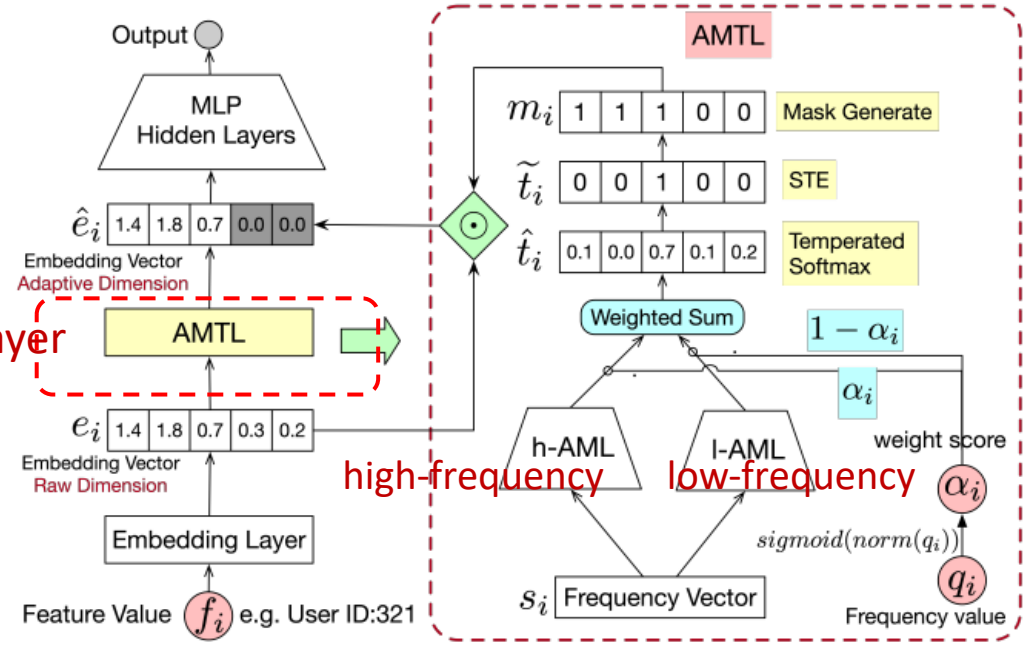
Embedding Size



Full Embedding Search-AMTL

- Search space: d^V (d is the embedding size and V is the vocabulary size)
- **Twins-based architecture** to avoid the unbalanced parameters update problem due to the different feature frequencies.
- The twins-based architecture acts as a frequency-aware policy network to search the optimal dimension for each feature value → relaxed to a continuous space by temperature softmax.

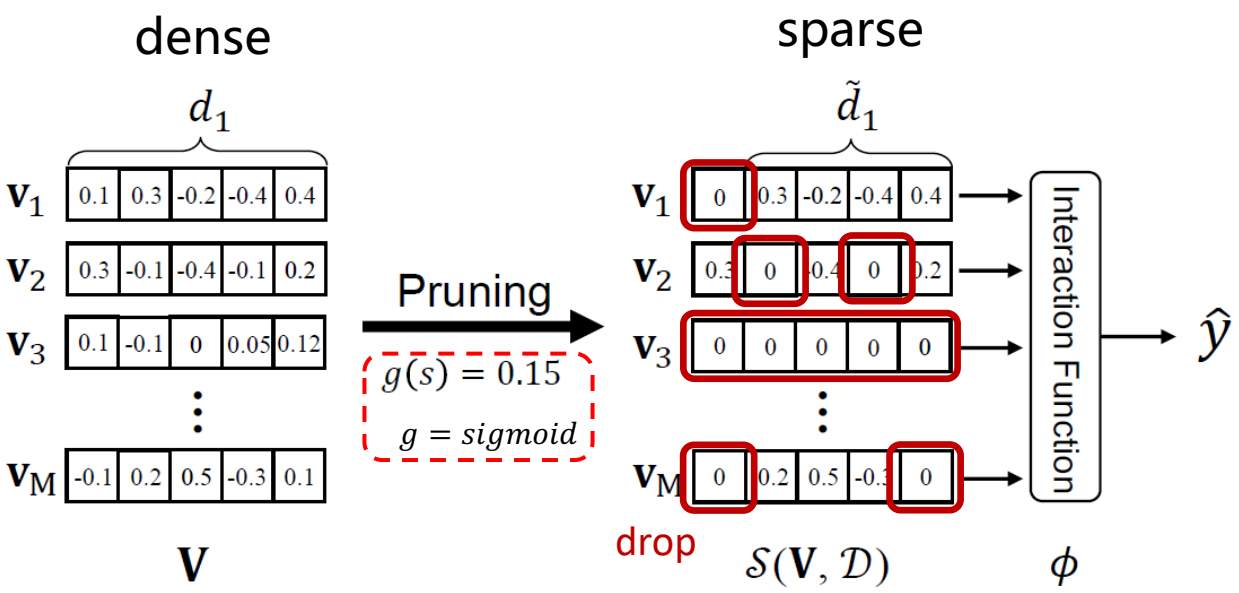




Learning effective and efficient embedding via an adaptively-masked twins-based layer. CIKM 2021.

Full Embedding Search-PEP

- Pruning-based Solution by enforcing **column-wise sparsity** on the embedding table with L_0 normalization.
- Search Space: 2^{Vd} (d is the embedding size and V is the vocabulary size)
- The learnable threshold can be jointly optimized with the model parameters via gradient-based back-propagation.



$$\min \mathcal{L}, s.t. \ ||\mathbf{V}||_0 \leq k, \quad \text{NP-hard}$$

Soft threshold re-parameterization：

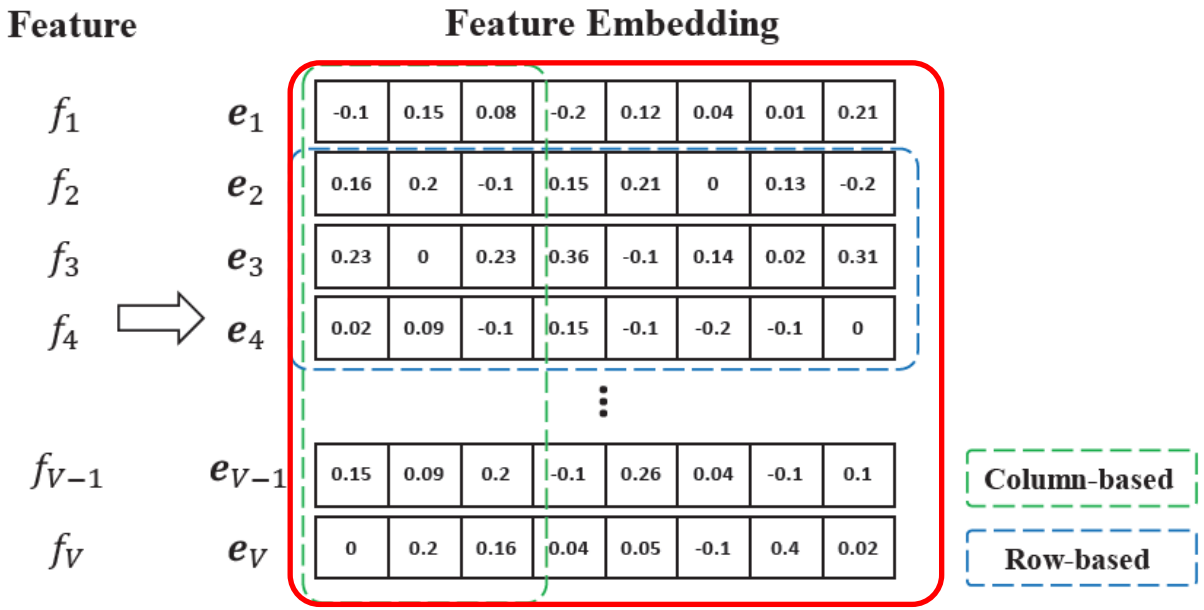
$$\hat{\mathbf{V}} = \mathcal{S}(\mathbf{V}, s) = sign(\mathbf{V})\text{ReLU}(|\mathbf{V}| - g(s)),$$

$$\min \mathcal{L}(\mathcal{S}(\mathbf{V}, s), \Theta, \mathcal{D}).$$

learnable pruning threshold

PEP: Learnable Embedding Sizes for Recommender Systems. ICLR 2021.

Full Embedding Search——Summary



- Full embedding search methods aim to search the optimal embedding dimension for each feature value, facing **huge search space** and impeding the search efficiency;
- To facilitate the search procedure, several approaches are proposed to **shrink the search space**, which can be categorized into three kinds: column-based, row-based, and column & row-based.

Column-based Embedding Search

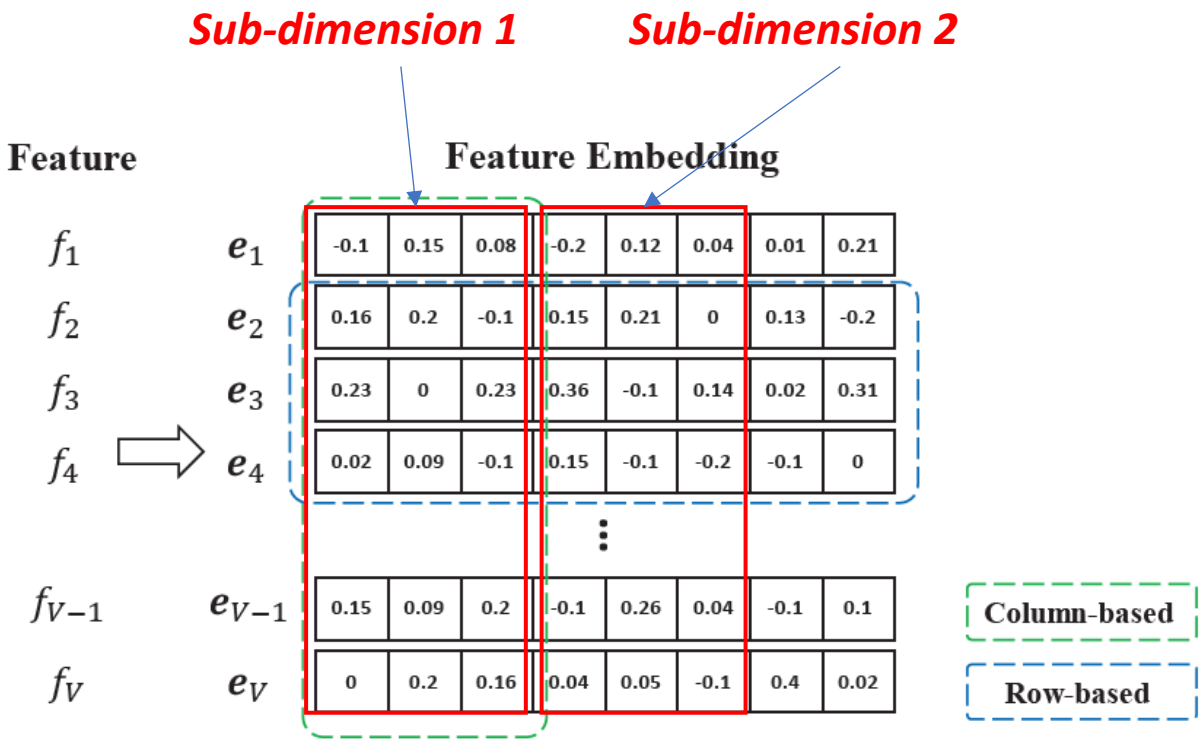
- The search space of Full Embedding Search (PEP and AMTL) is highly related with the embedding size d.
- To reduce the search space, AutoEmb and ESPAN divide the embedding dimension into several **column-wise sub-dimensions**.

Embedding Table

$$E \in R^{V \times d}$$

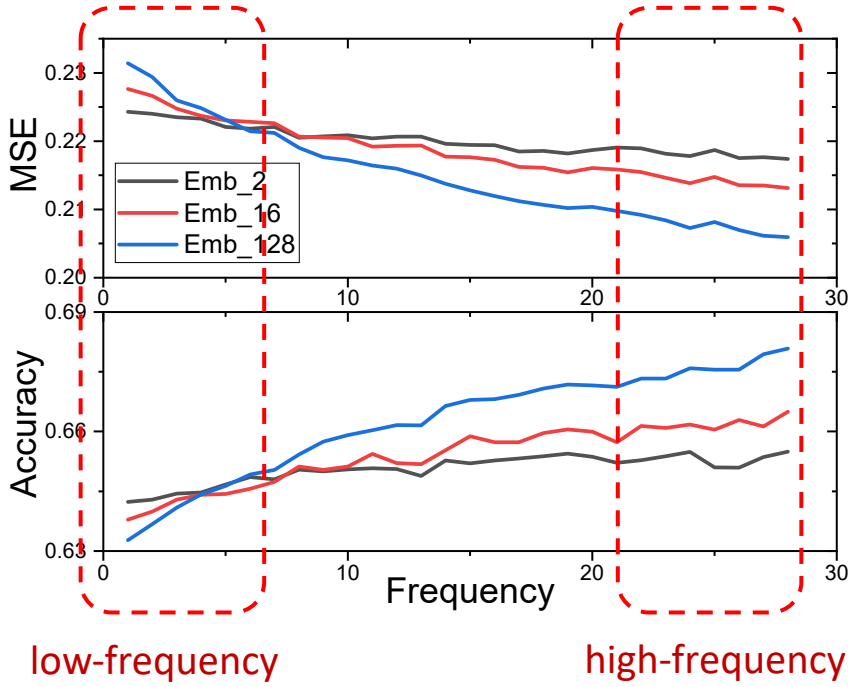
Number of Feature Values

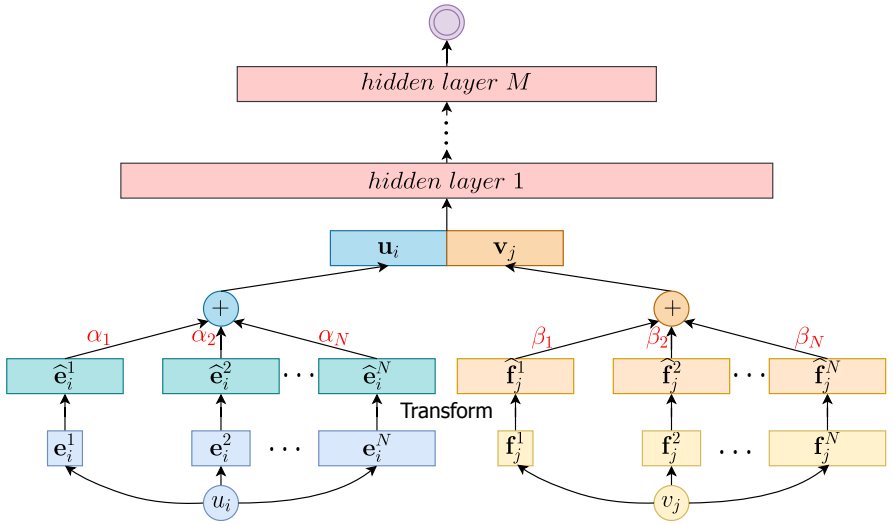
Embedding Size



Column-based Embedding Search-AutoEmb & ESPAN

- Reduce the search space by dividing the embedding dimension into several **candidate sub-dimensions**
- Dynamically search the embedding sizes for different **users and items**
  - ✓ Better performance
  - ✓ More efficient in memory

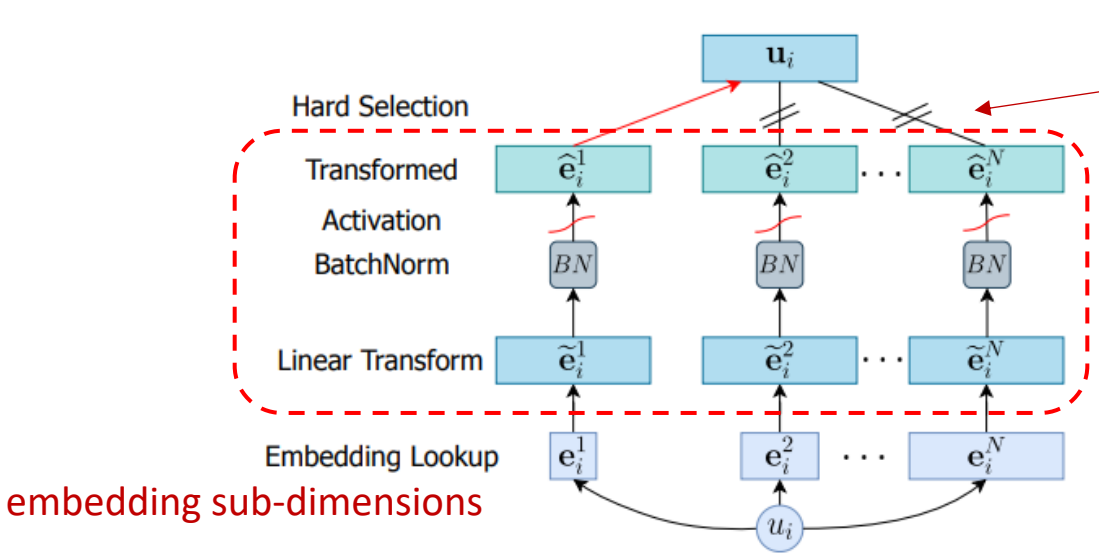




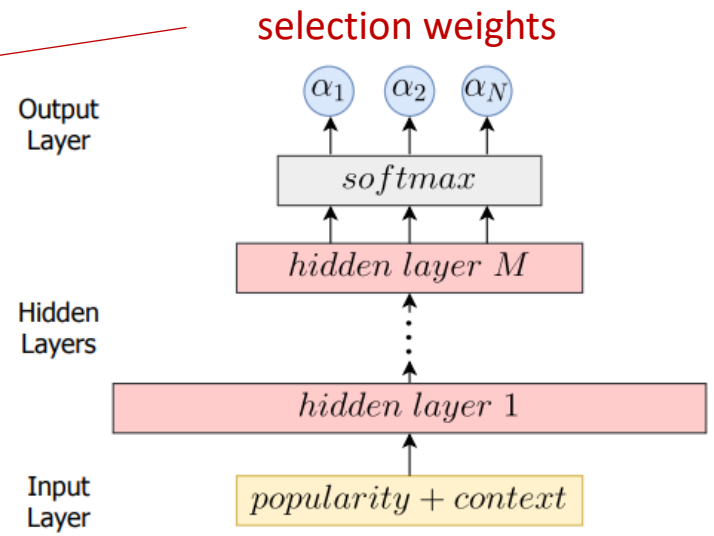
AutoEmb & ESPAN

Column-based Embedding Search-AutoEmb

- Search Space: From d^V to a^V (d is the embedding size and V is the vocabulary size, and a is **the number of sub-dimensions** for each feature)
- Two **controller networks** to decide the embedding sizes for users and items via end-to-end differentiable soft selection.
- Sum over the candidate sub-dimensions with learnable weights. **(Soft Selection)**
- The optimization is achieved by a **bi-level procedure**, where the controller parameters are optimized upon the validation set, while the model parameters are learned on the training set.



AutoEmb



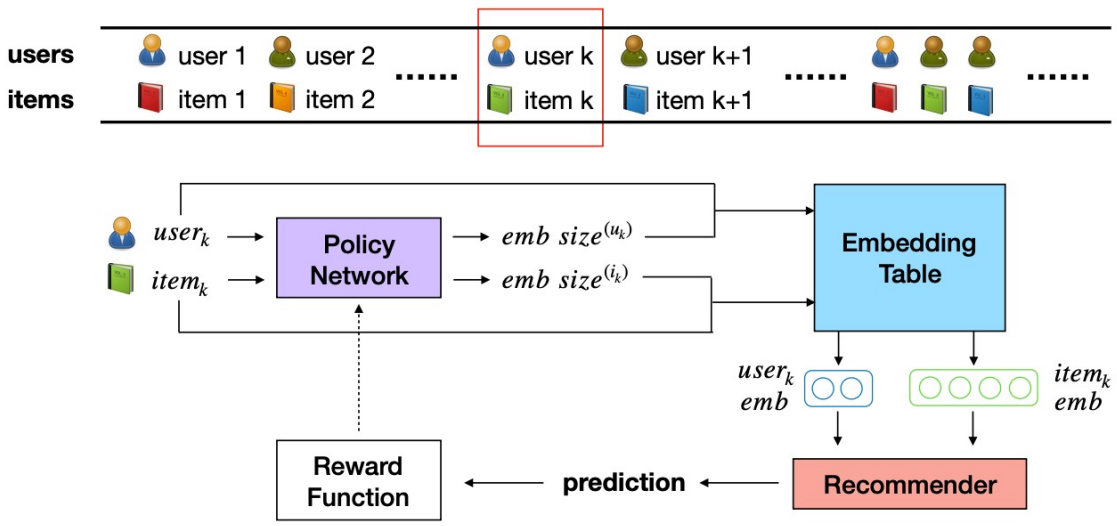
Controller network

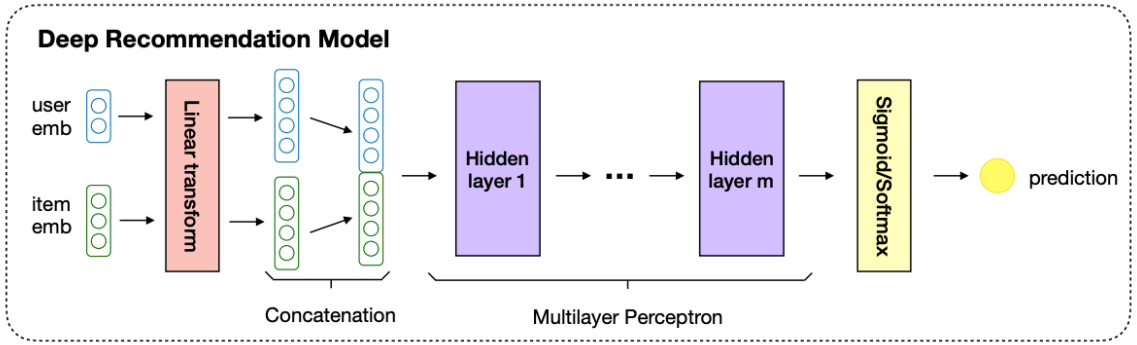
AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. ICDM 2021.

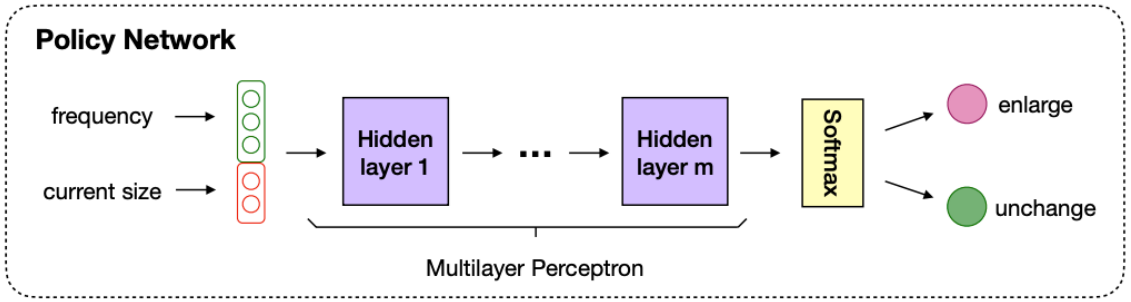
Column-based Embedding Search-ESPAN

- Embedding Size Adjustment Policy Network (ESAPN) - RL **(Hard Selection)**
- Policy network serves as RL agents for users and items, which adjusts the embedding sizes dynamically.
  - State: feature frequency and the current embedding size
  - Action: enlarge or unchanged the embedding size
  - Reward:

$$R^{(u)} = \frac{1}{T}\sum_{t=1}^{T} L_t^{(u)} - L$$

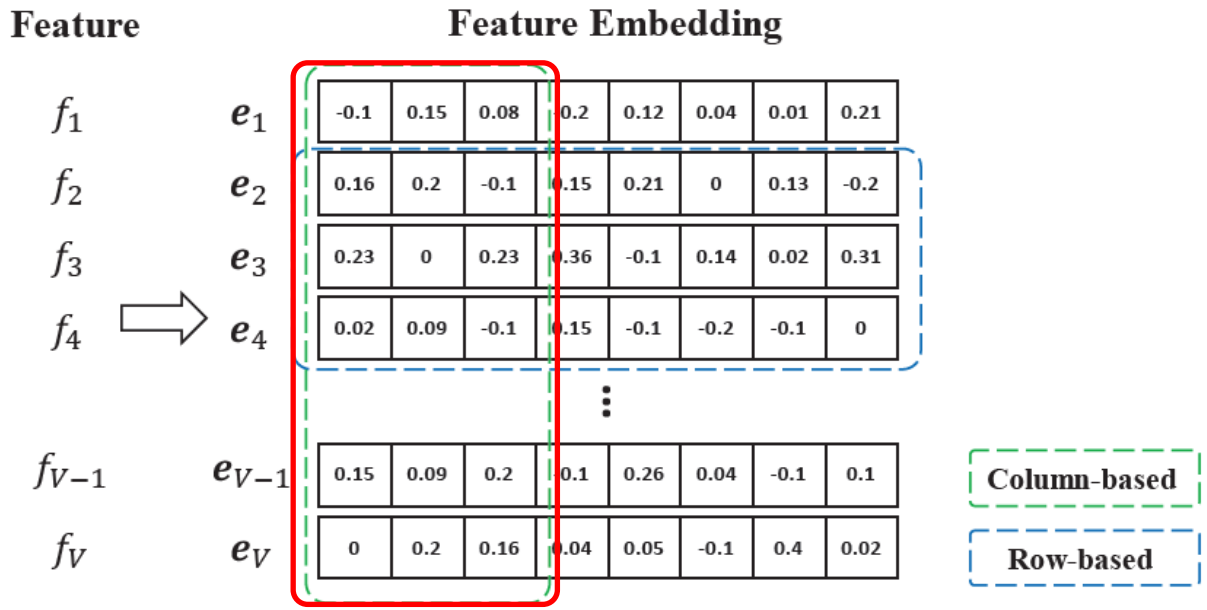






ESAPN: Automated Embedding Size Search in Deep Recommender Systems. SIGIR 2020.

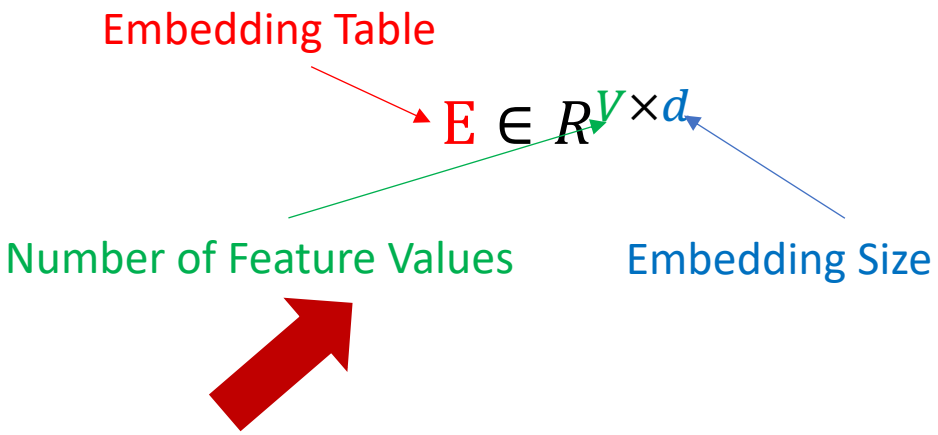
Column-based Embedding Search——Summary

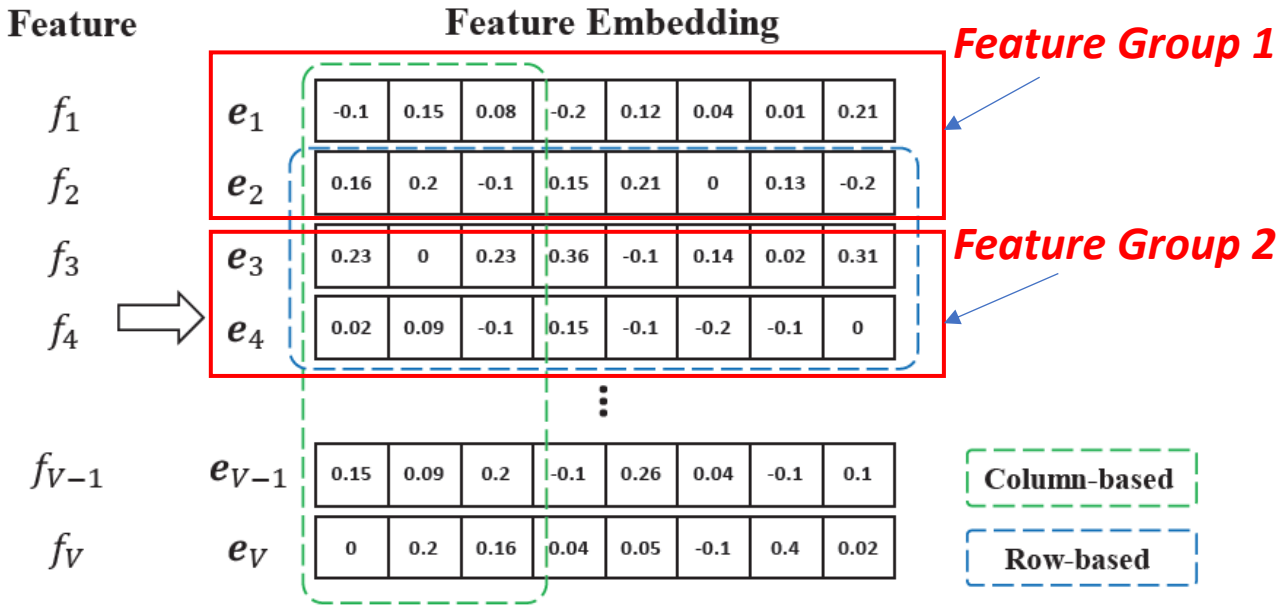


- Dividing the embedding dimension into column-wise sub-dimensions (e.g., AutoEmb, ESAPN) is conducive to **reducing the search space**;
- Using multiply embedding tables to generate several embedding vectors (e.g., AutoEmb, ESAPN) may incur obvious memory overhead, which can be avoid by shared-embeddings;
- Searching dimensions for each feature value will cause variable-length embedding vectors, which are **hard to store** in the fix-width embedding table and reduce memory.

Row-based Embedding Search

- AutoEmb and ESAPN shrink the search space by dividing the embedding dimension into candidate **column-wise sub-dimensions**.
- **Group** the feature values of a field based on some indicators (e.g., frequencies) and assign a **row-wise group embedding dimension** for all the values within the group.
- The search space is no longer related to the number of feature values, but to the number of pre-defined feature groups.
  - ✓ Shrink the search space, making it easier for the search algorithm to explore satisfactory results
  - ✓ Save the storage space physically



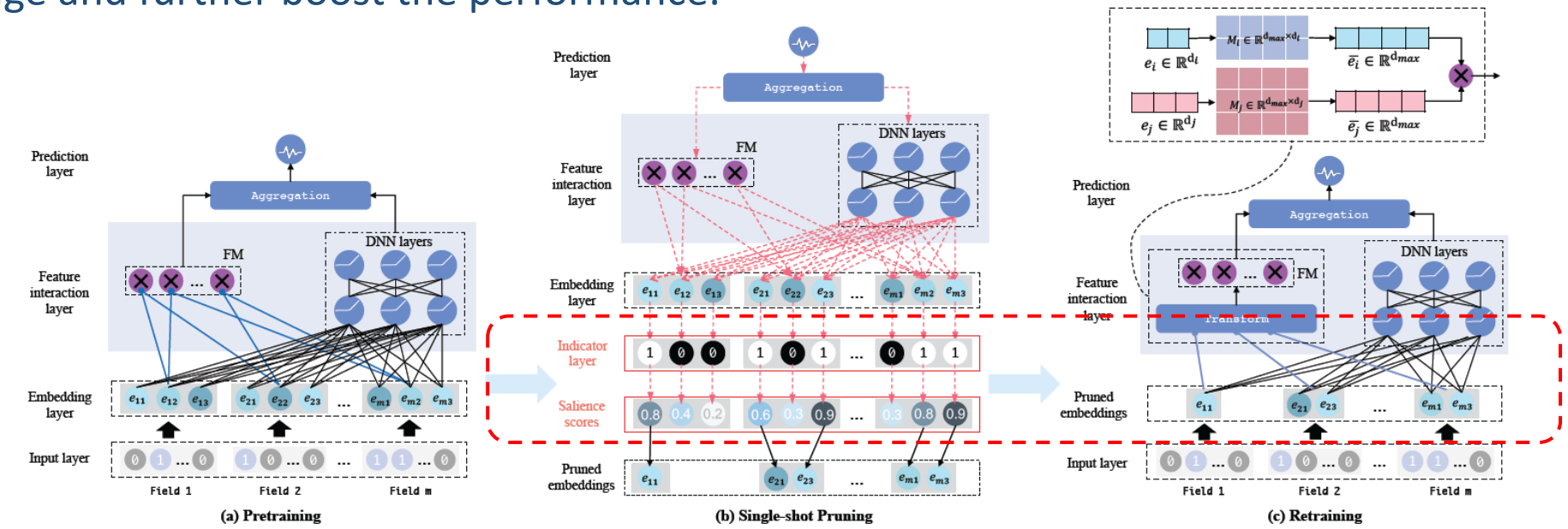


Row-based Embedding Search-SSEDS

- Set the number of groups for a feature field as b = 1 and search a global embedding dimension for all the feature values (**field-wise embedding dimension search**)
- Calculate the saliency scores for identifying the importance of each embedding dimension, which is measured by the change of the loss value

$$\Delta\mathcal{L}_{i,j} = \mathcal{L}(\hat{\mathbf{V}} \odot 1, \hat{\Theta}; \mathcal{D}) - \mathcal{L}(\hat{\mathbf{V}} \odot (1 - \epsilon_{i,j}), \hat{\Theta}; \mathcal{D}) \qquad \epsilon_{i,j} \in \{0,1\}^{\sum_i^m n_i \times d}$$

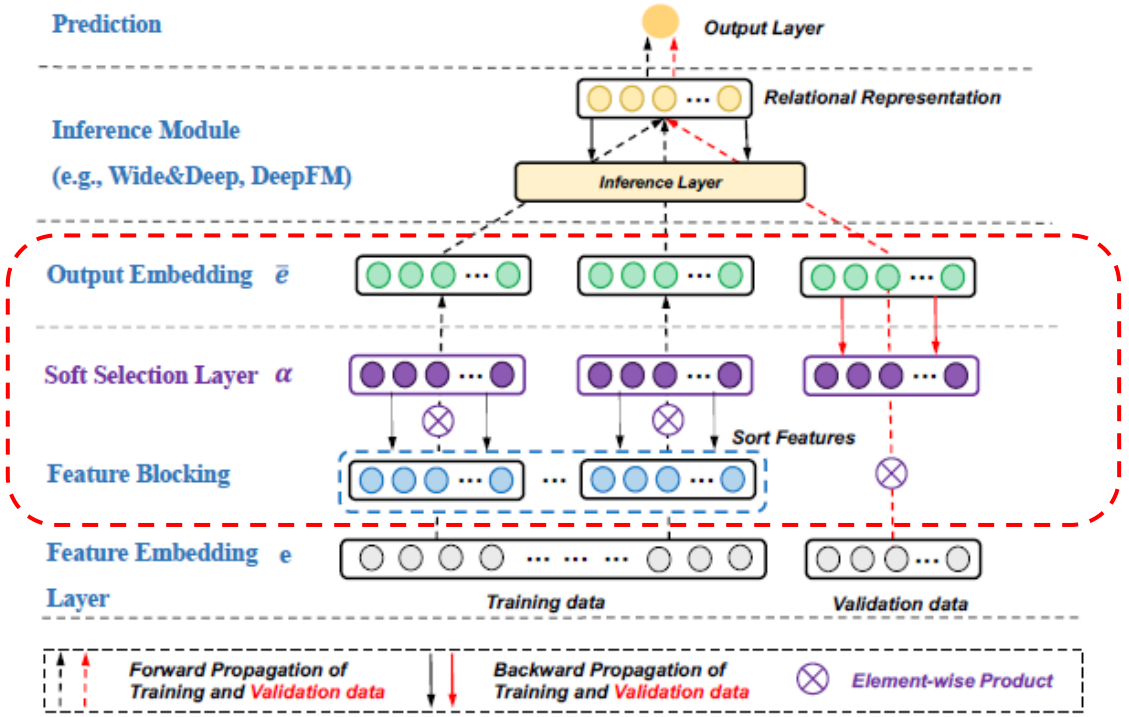
- Top-k scores can be retained according to the memory budget and the model will be retrained to save storage and further boost the performance.



Single-shot Embedding Dimension Search in Recommender System. SIGIR 2022.

Row-based Embedding Search-AutoSrh

- To balance the search efficiency and performance, AutoSrh splits the features into **multi-groups** based on the ***feature frequencies*** *or* ***clustering***.
- Feature blocking stage —— search space: from 2^{Vd} into 2^{bd} (where b is **the number of groups**)
- Soft selection layer to identify the importance of each dimension in the feature embedding
- To relax the search space to be continuous during the search stage, a gradient-based bi-level optimization procedure is proposed

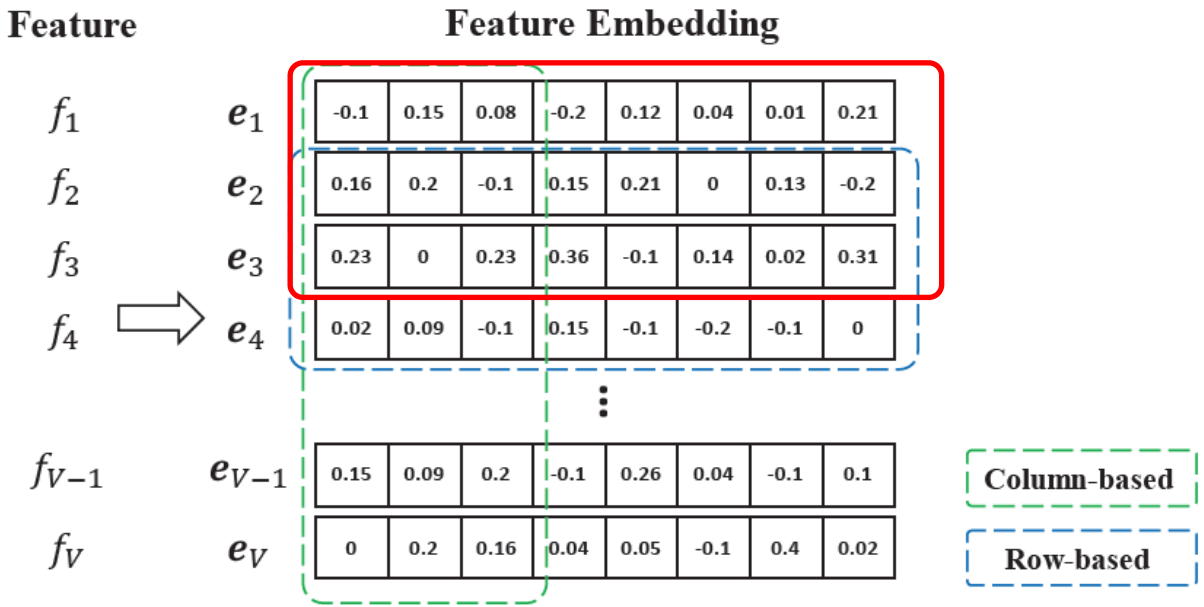


Search stage: $\tilde{\mathbf{e}}_i = \mathbf{e}'_i \odot \boldsymbol{\alpha}_{l*}$

Derive stage: $\mathbf{E}'_{i,j} = \begin{cases} 0, & \text{if } |\tilde{\mathbf{E}}_{i,j}| < \epsilon \\ \tilde{\mathbf{E}}_{i,j}, & \text{otherwise} \end{cases}$ pre-defined threshold

AutoSrh: An Embedding Dimensionality Search Framework for Tabular Data Prediction. TKDE 2022.

Row-based Embedding Search——Summary

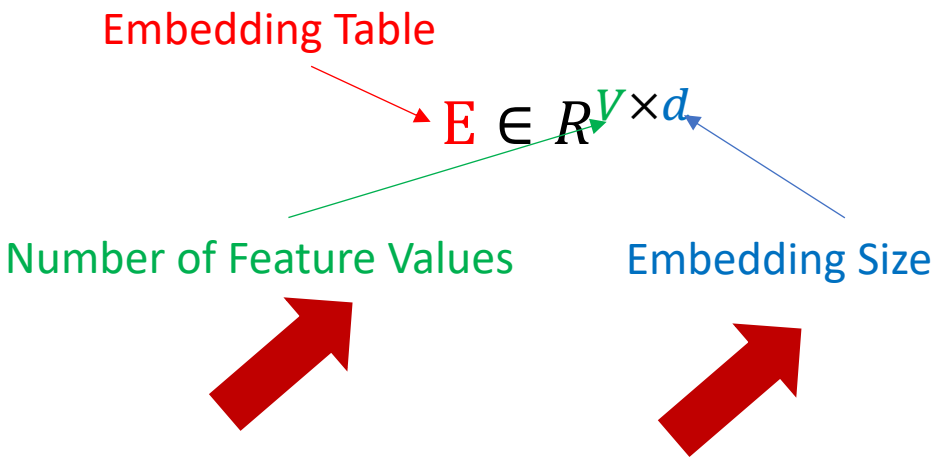


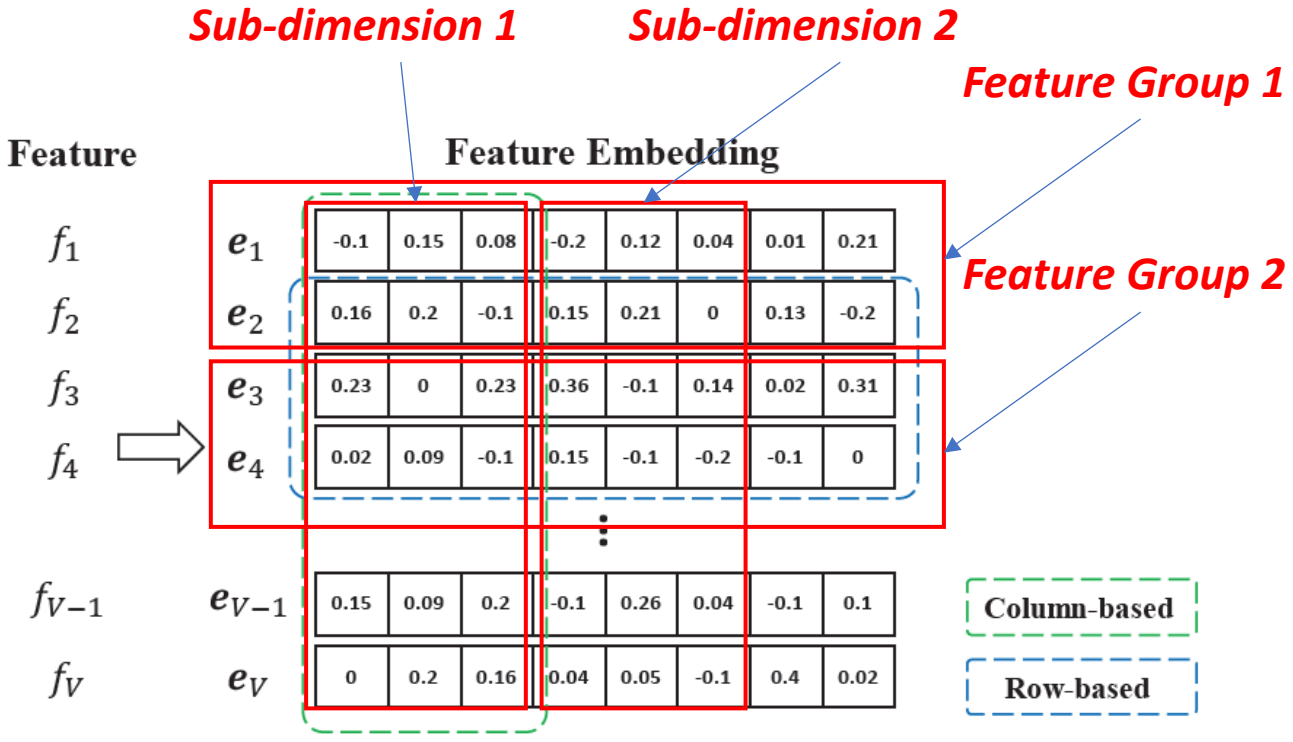
- Row-based embedding search methods explore optimal embedding dimension for a group of feature values, **shrinking the search space**;
- In comparison with the column-based embedding search methods, row-based search methods conduces to **truly saving memory** because feature values within a group are assigned with a same embedding dimension, which can be stored in a fix-width embedding table.

Column & Row-based Embedding Search

- The column-based and row-based methods make different assumptions to reduce the search space from different perspective.
- To further improve the search efficiency, several works combine these two methods and reduce the search space significantly.

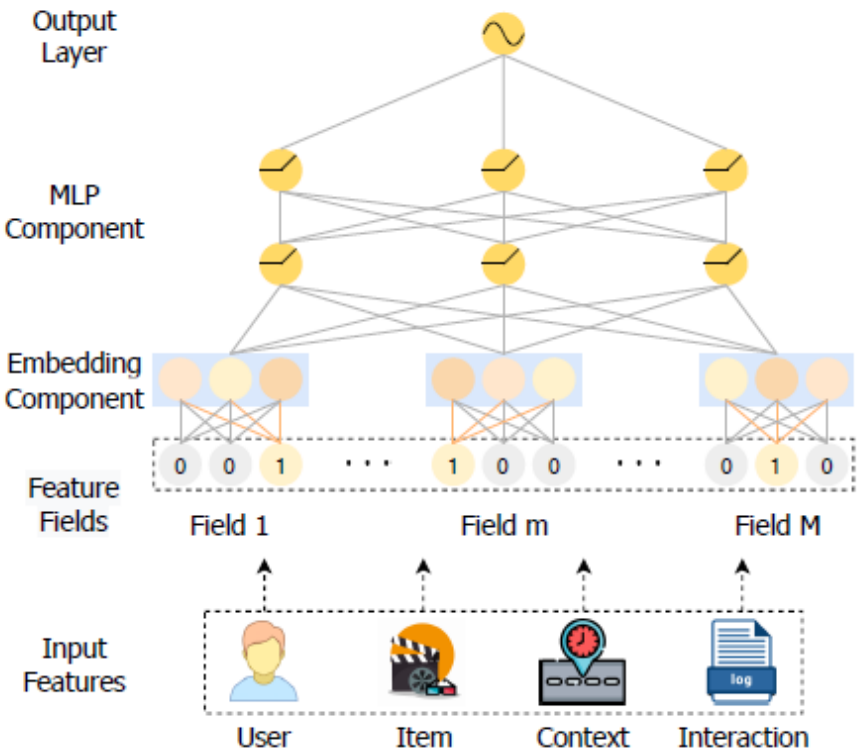
column-wise sub-dimensions
row-wise group embedding dimension





Column & Row-based Embedding Search-AutoDim

- Pre-define several candidate sub-dimensions like AutoEmb. *(column-wise)*
- Set the number of groups b = 1 and search a global embedding dimension for **all the feature values of the field**, like SSEDS. *(raw-wise)*
- Search Space: a^{m} (a is **the number of sub-dimensions** for each feature, m is **the number of feature fields**)



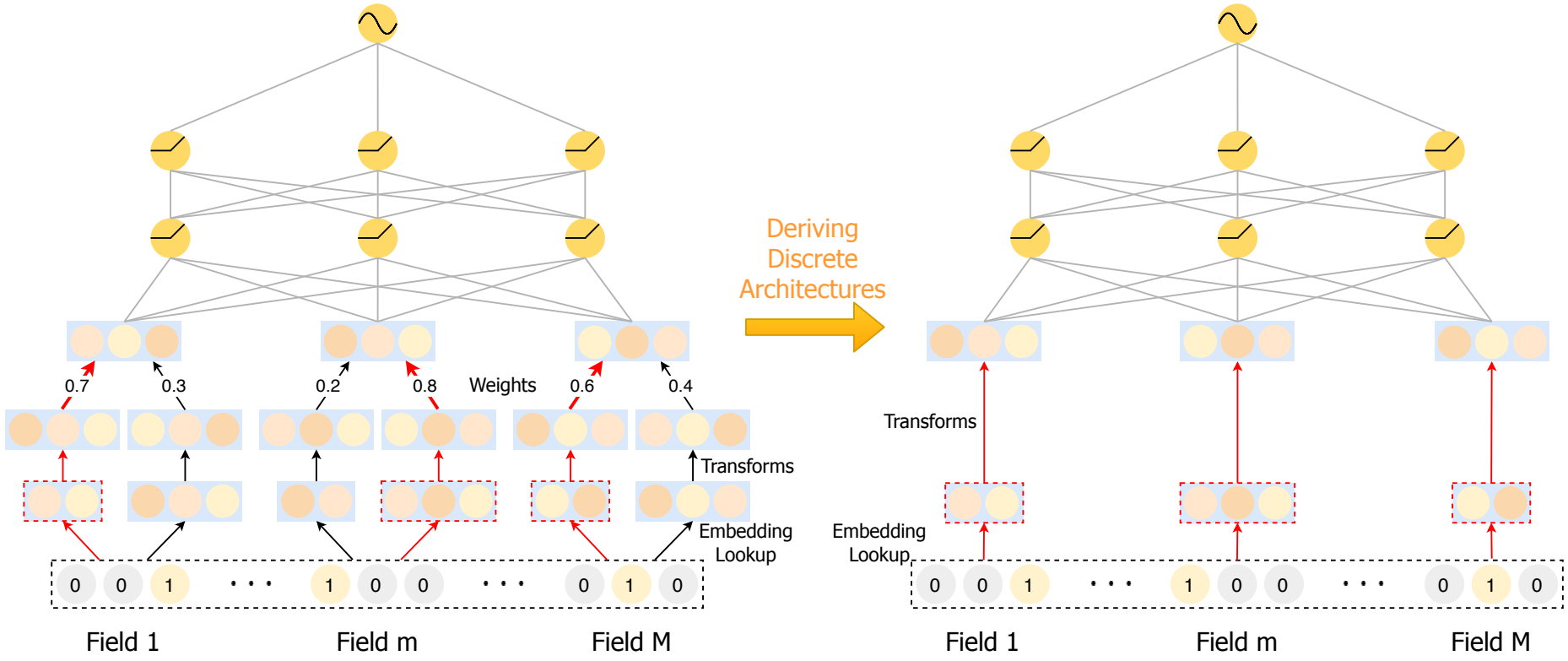
Goal:

Select optimal embedding dimensions for **different feature fields** automatically in a data-driven manner.

AutoDim: Field-aware Embedding Dimension Search in Recommender Systems. WWW 2021.

Column & Row-based Embedding Search-AutoDim

Two-stage framework



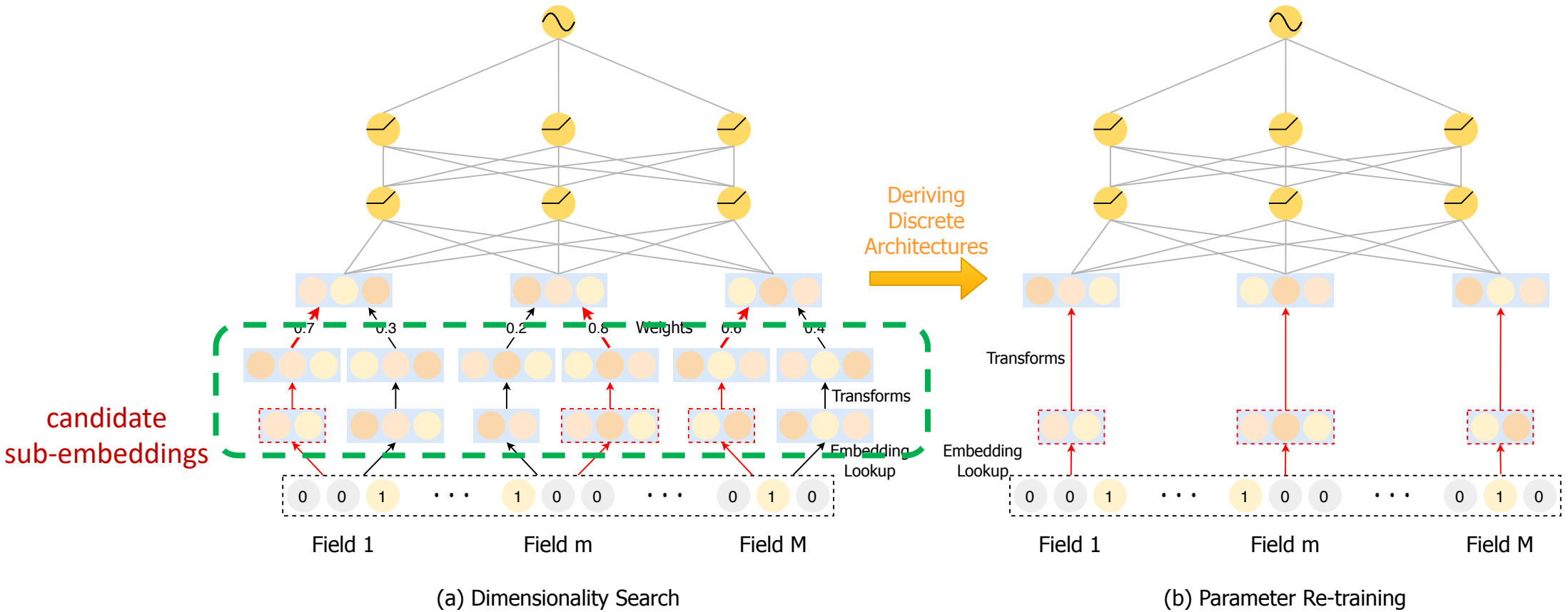
(a) Dimensionality Search

(b) Parameter Re-training

- Dimensionality search stage: find the optimal embedding dimension for each feature field
- Parameter re-training stage: select the optimal embedding dimension and re-train the model parameters

Column & Row-based Embedding Search-AutoDim

Dimensionality search stage



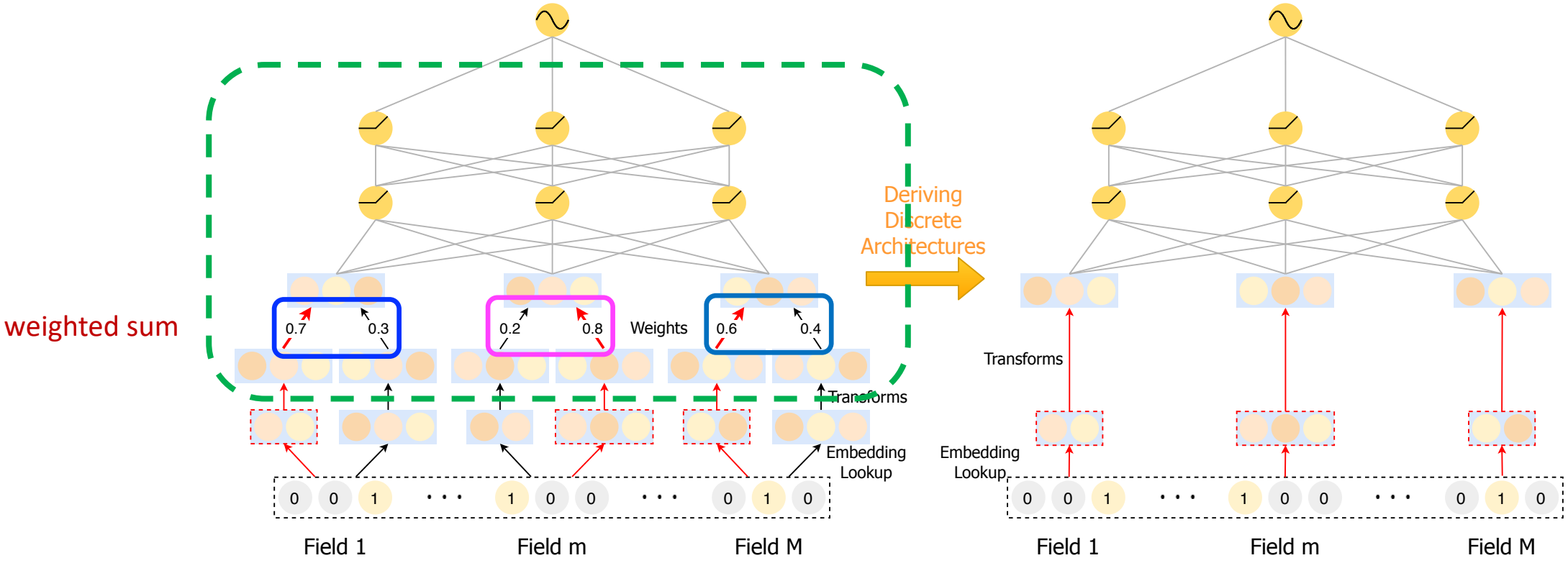
(a) Dimensionality Search

(b) Parameter Re-training

- Transform layer: map the embeddings into a same dimensions
- Batch Normalization layer: unify the scales

Column & Row-based Embedding Search-AutoDim

Dimensionality search stage



(a) Dimensionality Search

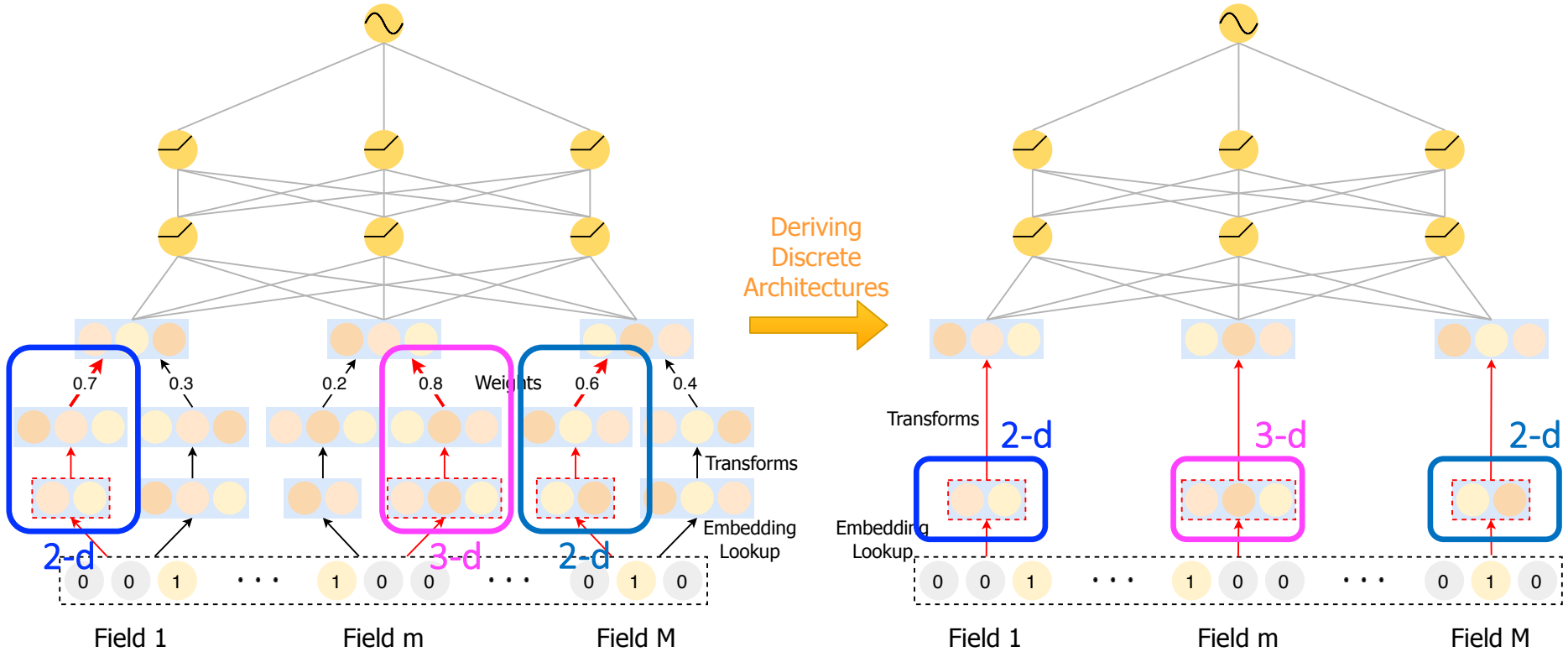
(b) Parameter Re-training

AutoDim searches the dimensions in a soft and continuous fashion via the **Gumbel Softmax**

- The architecture weights are optimized upon the validation set
- Others model parameters are learned upon the training set

Column & Row-based Embedding Search-AutoDim

Parameter re-training stage



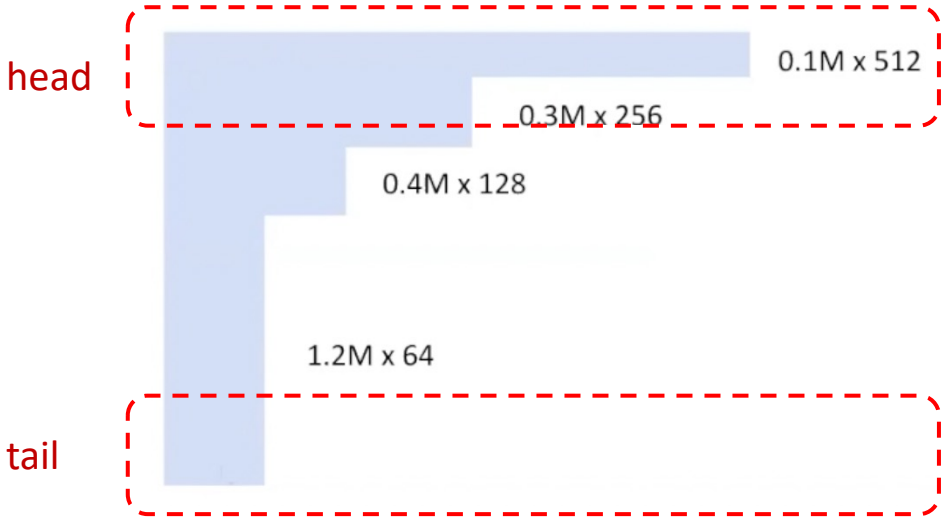
(a) Dimensionality Search

(b) Parameter Re-training

- The optimal embedding with the largest weight is selected for each feature field
- Retrain the model parameters to obtain the final model

Column & Row-based Embedding Search-NIS

- NIS also reduces the search space from both row-wise and column-wise perspectives



Head Feature

- More data, more information
- larger embedding size is reasonable

Tail Feature

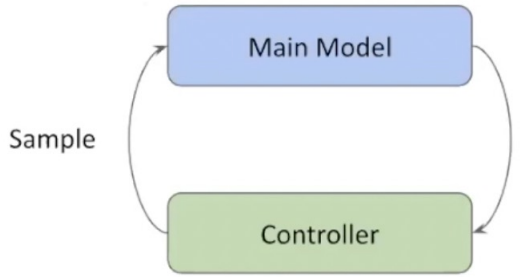
- Less data, less information
- Small embedding size is enough

NIS: Neural Input Search for Large Scale Recommendation Models. KDD 2020.

Column & Row-based Embedding Search-NIS

RL-based AutoML solution

- Main model is the deep recommendation model
- Controller learns to sample embedding dimensions that generate higher reward.
- Reward: $R = R_Q - \lambda * C_M$



optimization objective
training cost

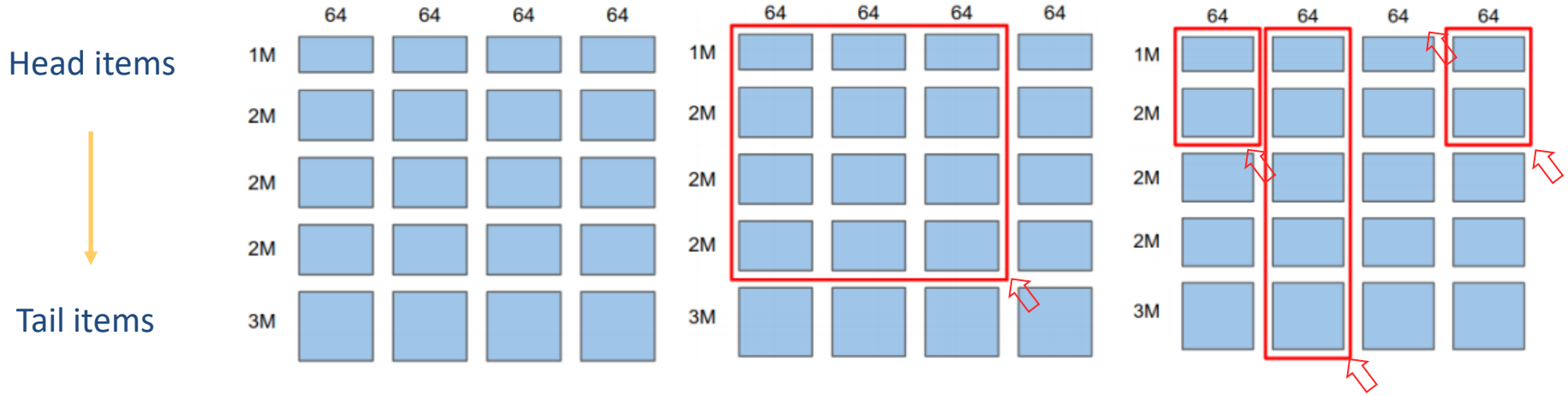
$$\sum_{F \in \mathcal{F}} v_F \times d_F \leq \mathcal{C}$$

$$\sum_{F \in \mathcal{F}} \sum_{i=1}^{M_F} v_{F_i} \times d_{F_i} \leq \mathcal{C}$$

multiple embedding blocks

one choice **(7M, 192)**

a sequence of choices **[(3M, 192), (7M, 64) …]**

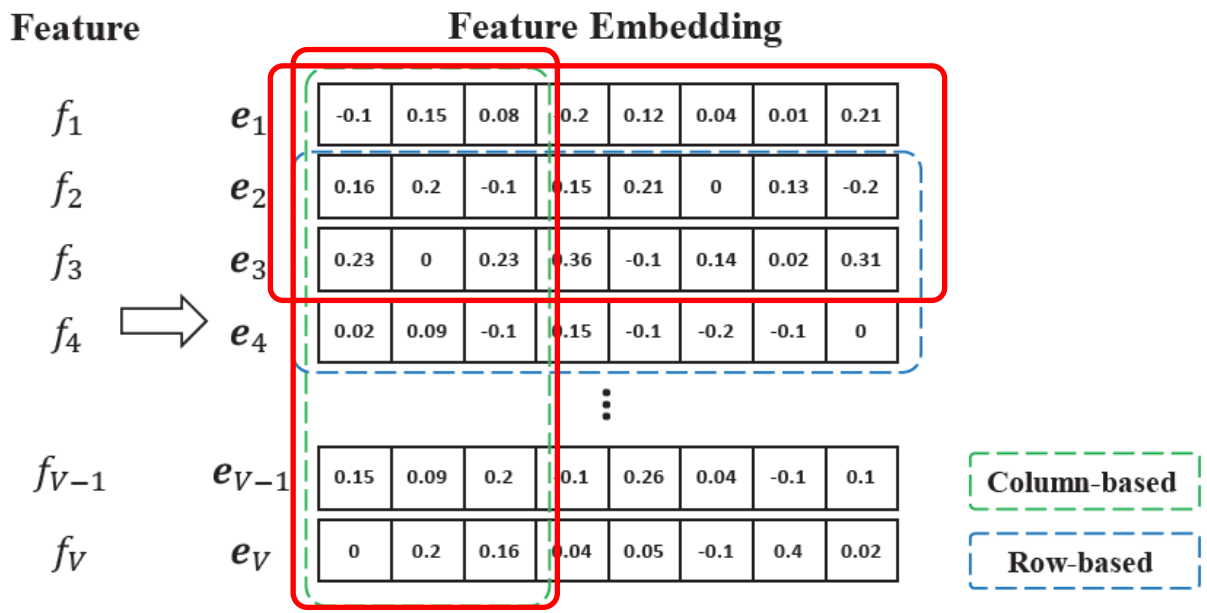


Single-size Embedding (SE)

Multi-size Embedding (ME)

Embedding Blocks: discretizing an embedding matrix of size v × d into S × T sub-matrices

Column & Row-based Embedding Search——Summary



- Although it is theoretically optimal to search the suitable dimension for each feature value, it poses great challenges to efficient search algorithm. Instead, shrinking the search space in an appropriate manner may result in better performance;
- Reducing the search space from both row-wise and column-wise perspectives attributes to reducing the search space and achieving better results;
- The evolution of search space **from detailed to abstract** can lead to higher efficiency.

Combination-based Embedding-AutoDis

Besides searching embedding dimension dynamically for each features, learning embedding via **combination** is also a trend for feature representation learning

Existing methods for numerical feature representation have some **limitations**:

1. Category 1: No Embedding

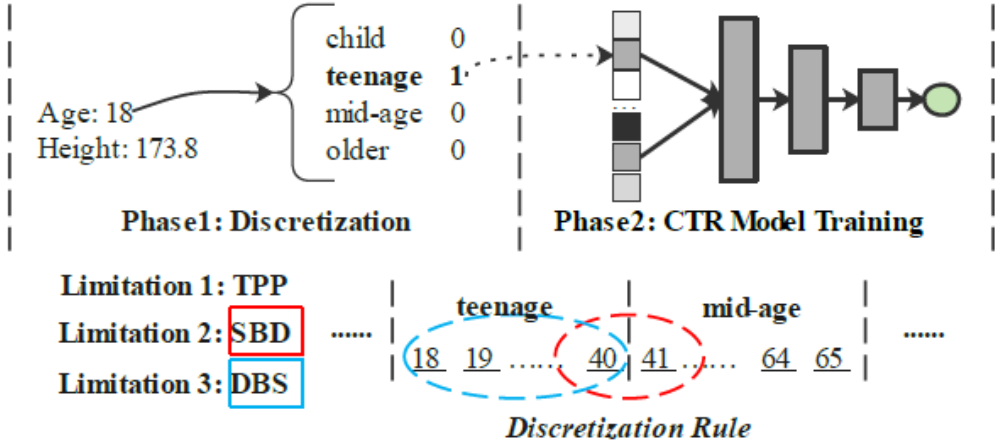
- **Low capacity**: difficult to capture informative knowledge of numerical fields.
- **Poor compatibility**: difficult to adapt to some models (e.g., FM).

2. Category 2: Field Embedding

- **Low capacity**: single uniform field-specific embedding.

3. Category 3: Discretization

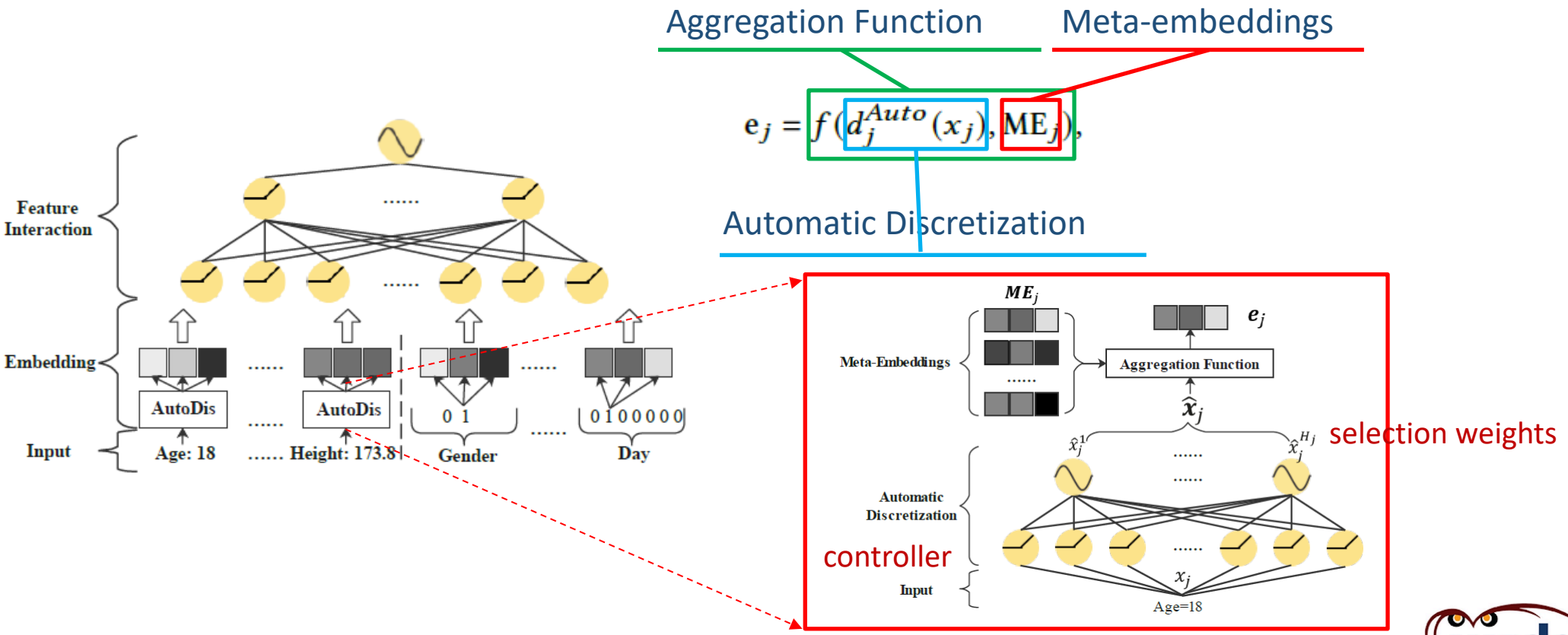
- **TPP** (Two-Phase Problem)
- **SBD** (Similar value But Dis-similar embedding)
- **DBS** (Dis-similar value But Same embedding)



AutoDis: An Embedding Learning Framework for Numerical Features in CTR Prediction. KDD 2021.

Combination-based Embedding-AutoDis

AutoDis is a numerical features embedding learning framework with **high model capacity, end-to-end training and unique representation** properties preserved.



Summarize DRS Embedding

Method	Column-Wise	Row-Wise	Search Space	Multi-Embedding	Search Algorithm	Memory Reduction
AMTL	×	×	d^V	×	Gradient	×
PEP	×	×	2^{Vd}	×	Regularization	×
AutoEmb	√	×	d^V	√	Gradient	×
ESAPN	√	×	a^V	√	RL	×
SSEDS	×	√	2^{md}	×	Gradient	√
AutoSrh	×	√	2^{bd}	×	Gradient	√
AutoDim	√	√	a^m	×	Gradient	√
NIS	√	√	ab or b^a	×	RL	√
RULE	√	√	2^{ab}	×	Evolutionary	√
ANT	-	-	2^{kV}	×	Gradient	√
AutoDis	-	-	2^{kV}	×	Gradient	√

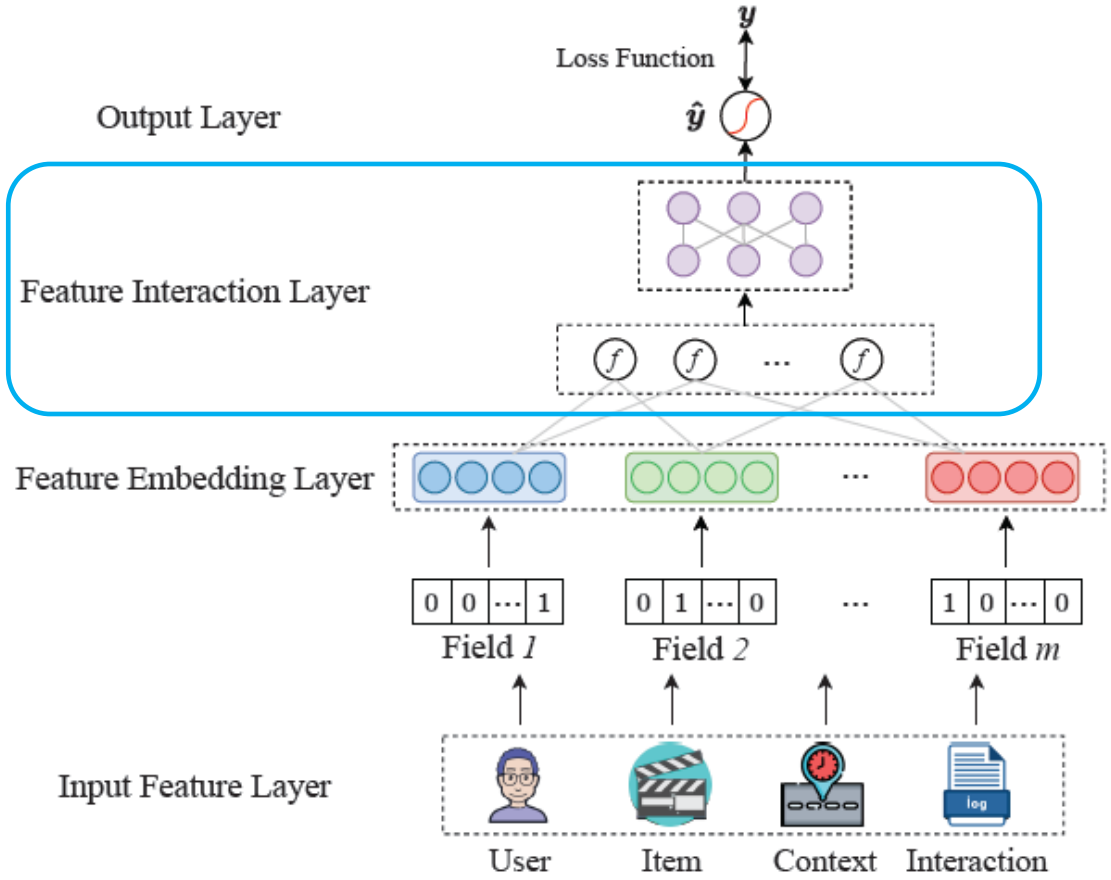
* d is the embedding size, V is the vocabulary size, m is the number of feature fields, a is the number of sub-dimensions, b is the number of feature groups, k is the number of meta-embeddings. (a < d, b<<V)

Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
- DRS Interaction Components
  - Feature Interaction Search
  - Interaction Function Search
  - Interaction Block Search
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

Background

Effectively modelling **feature interactions** is important.



- Both low-order and high-order feature interactions play important roles to model user preference.
  - People like to download popular apps → id of an app may be a signal
  - People often download apps for food delivery at meal time → interaction between app category and time-stamp may be a signal
  - Male teenagers like shooting game → interaction of app category, user gender and age may be a signal
- Most feature interactions are hidden in data and difficult to identify (e.g., “diaper and beer" rule)

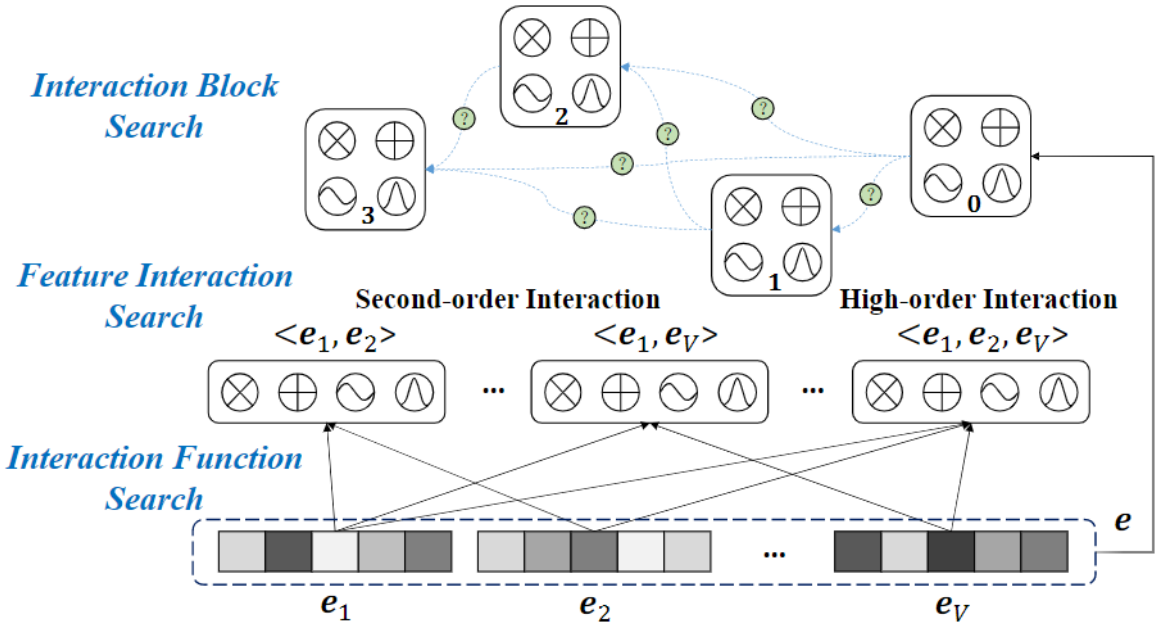
Background

The challenges of modelling **feature interactions**:

1) Enumerate all feature interactions
    - Large memory and computation cost
    - Difficult to be extended into high-order interactions
    - Useless interactions
2) Require human efforts to identify important **feature interactions**
    - High labor cost
    - Risks missing some counterintuitive (but important) interactions
3) Require human efforts to select appropriate **interaction functions**
    - Human expert knowledge
    - Global interaction function for all the feature interactions

Background

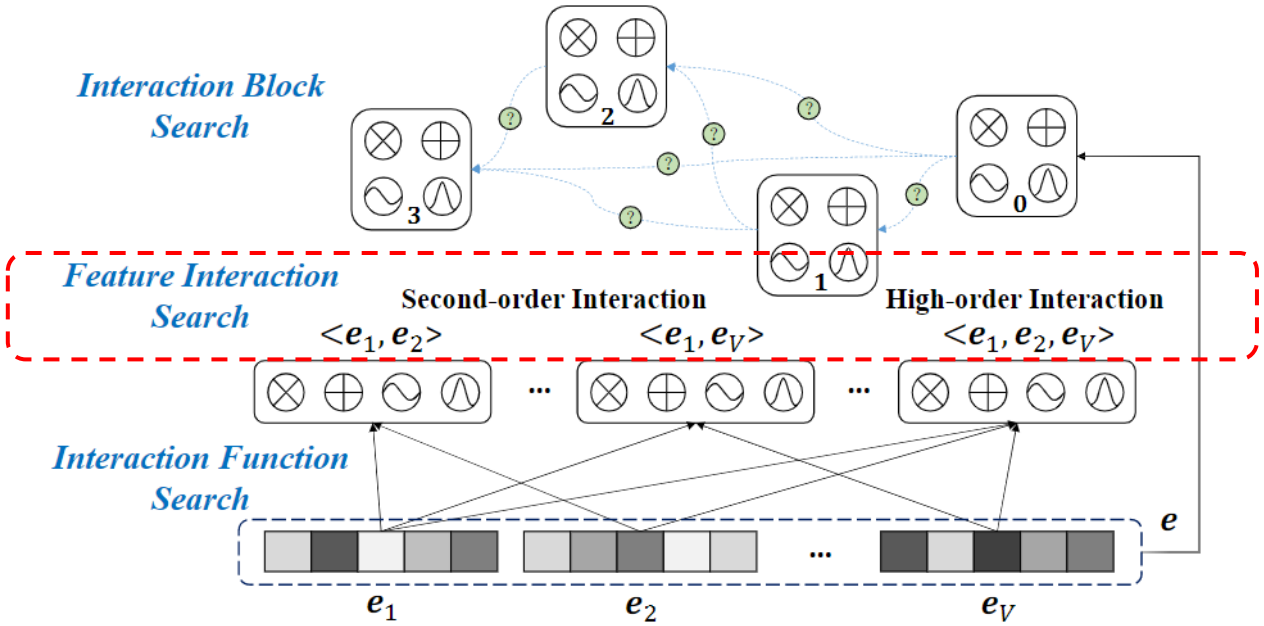
Automatically **select important feature interactions with appropriate interaction functions**



AutoML for feature interaction search:

1. Feature Interaction Search —— search beneficial feature interactions
2. Interaction Function Search —— search suitable interaction functions
3. Interaction Block Search —— search operations over the whole representation

Feature Interaction Search

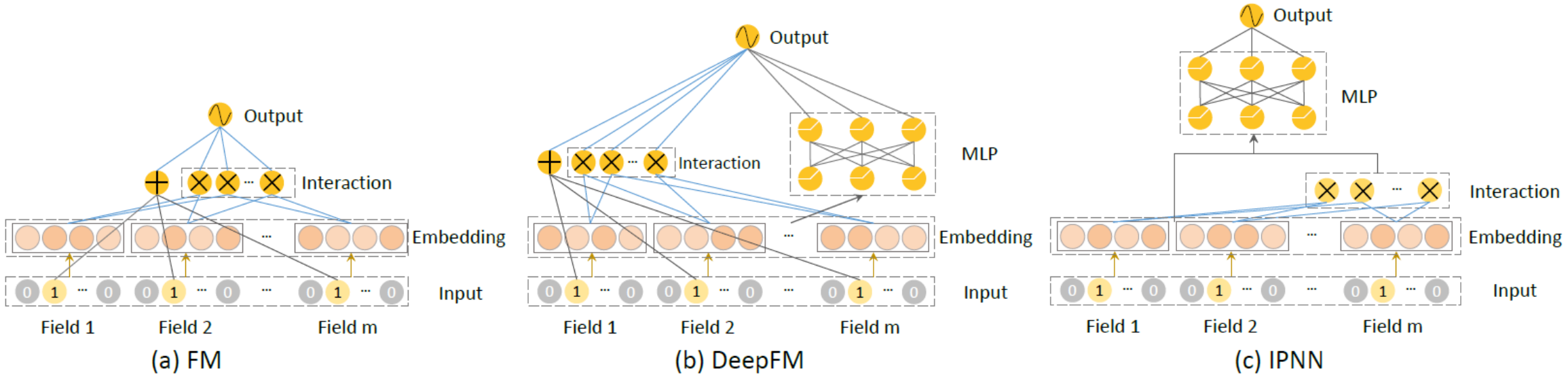


AutoML for feature interaction search:

1. **Feature Interaction Search —— search beneficial feature interactions**
2. Interaction Function Search —— search suitable interaction functions
3. Interaction Block Search —— search operations over the whole representation

Feature Interaction Search-AutoFIS

- Not all the feature interactions are useful.
- Identify such noisy feature interactions and filter them.



AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. KDD 2020

Feature Interaction Search-AutoFIS

- Search Stage
  - Detect useful feature interactions

- Retrain Stage
  - Retrain model with selected feature interactions

Feature Interaction Search-AutoFIS

Search Stage:

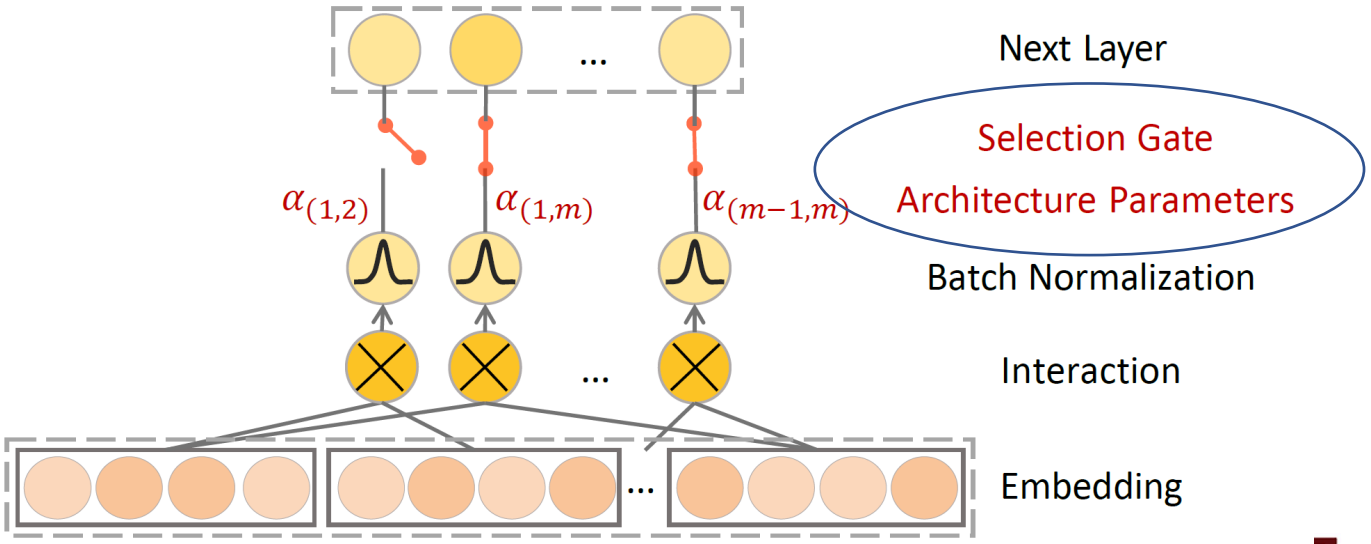
- Gate for each feature interaction
  - Huge search space $2^{C_m^2}$ (m is the number of feature field)
- To make such process differentiable, AutoFIS relaxes the discrete search space to be continuous, by defining architecture parameters α.
  - Batch Normalization to eliminate scale coupling
  - Using GRDA Optimizer to obtain stable and sparse architecture parameters

$$l_{\text{AutoFIS}} = \langle w, x \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \alpha_{(i,j)} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$$

Indicator α = 0 or 1

Retrain Stage:

- Abandon unimportant feature interactions
- Retrain model



Feature Interaction Search-AutoGroup

The limitation of AutoFIS:

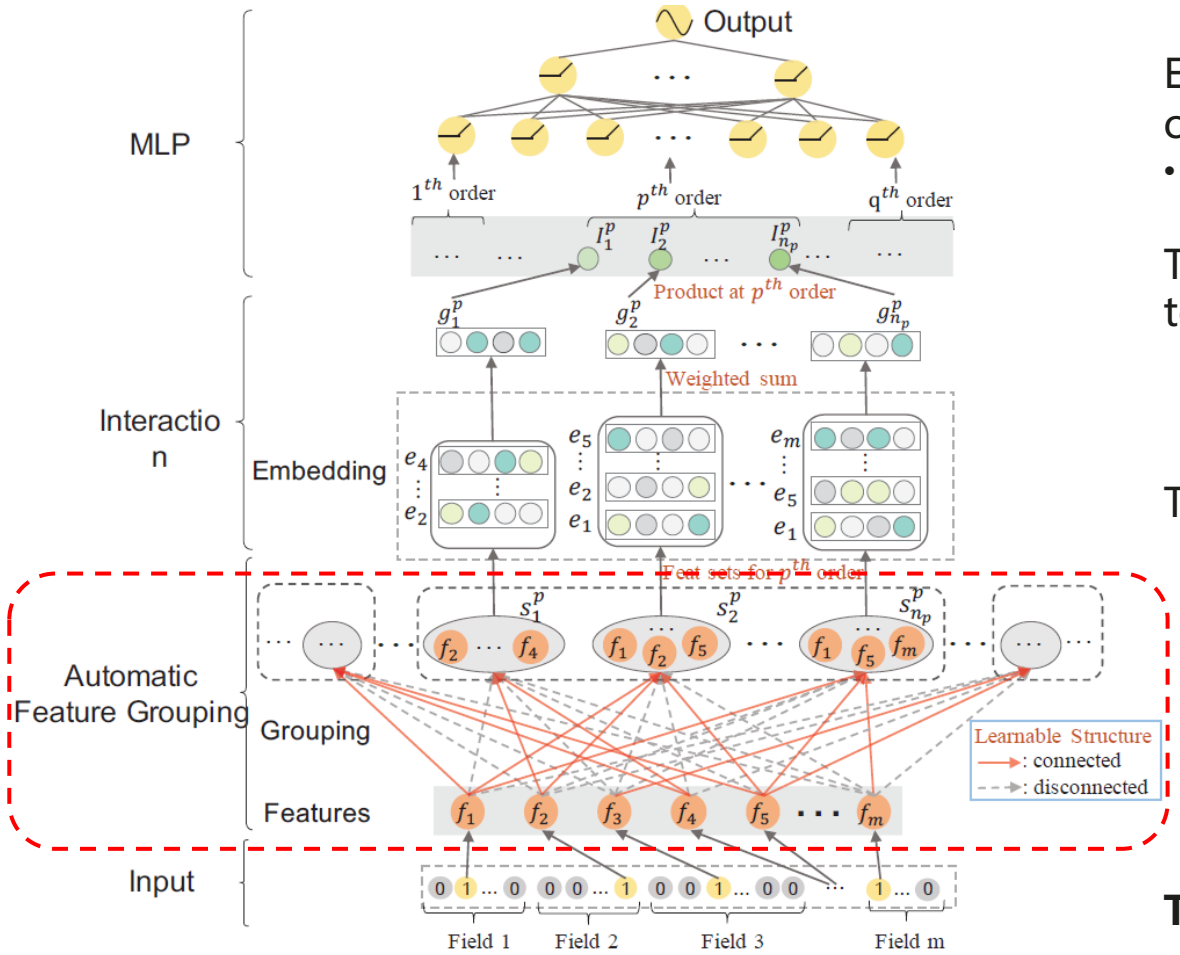
- When searching **high-order feature interactions**, the search space of AutoFIS is huge, resulting in low search efficiency.

Solution of AutoGroup:

- To solve the ***efficiency-accuracy dilemma***, AutoGroup proposes automatic feature grouping, reducing the pth-order search space from $2^{\mathrm{C}_{\mathrm{m}}^{\mathrm{p}}}$ to 2^{gm} (g is the number of pre-defined groups)

Feature Interaction Search-AutoGroup

Feature Grouping Stage:



Each feature is possible to be selected into the feature sets of each order.

- $\Pi_{i,j}^{p} \in \{0,1\}$: whether select feature f_i into the j^{th} set of order-p.

To make the selection differentiable, we relax the binary discrete value to a softmax over the two possibilities:

$$\overline{\Pi}_{i,j}^{p} = \frac{1}{1+\exp(-\alpha_{i,j}^{p})}\Pi_{i,j}^{p} + \frac{\exp\left(-\alpha_{i,j}^{p}\right)}{1+\exp\left(-\alpha_{i,j}^{p}\right)}(1-\Pi_{i,j}^{p}).$$

To learn a less-biased selection probability, we use Gumbel-Softmax:

$$\left(\overline{\Pi}_{i,j}^{p}\right)_o = \frac{\exp(\frac{\log \alpha_o + G_o}{\tau})}{\sum_{o' \in \{0,1\}} \exp(\frac{\log \alpha_{o'} + G_{o'}}{\tau})} \text{ where } o \in \{0,1\}.$$

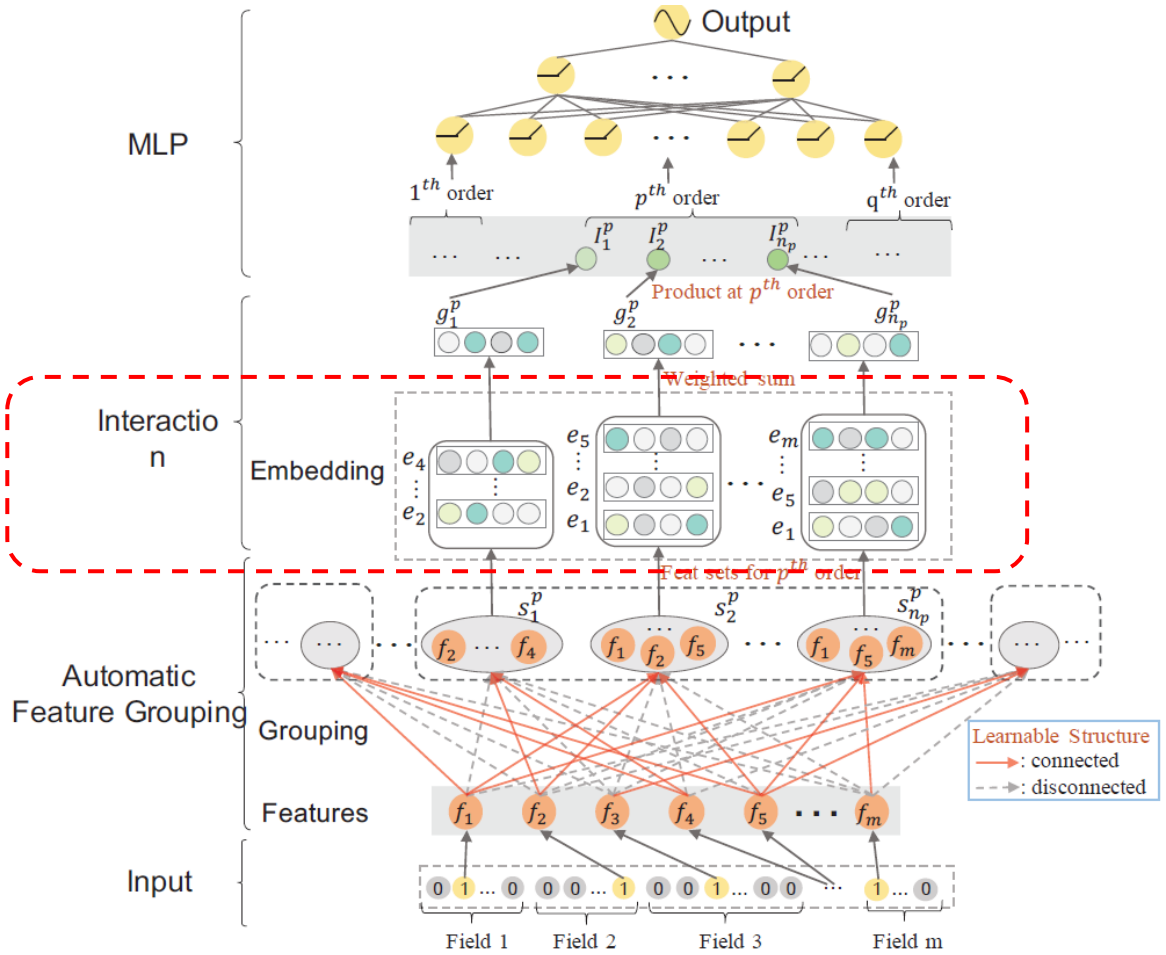
$$\alpha_0 = \frac{1}{1+\exp(-\alpha_{i,j}^{p})} \qquad \alpha_1 = \frac{\exp\left(-\alpha_{i,j}^{p}\right)}{1+\exp\left(-\alpha_{i,j}^{p}\right)}$$

$$G_o = -\log(-\log u) \text{ where } u \sim Uniform(0,1)$$

Trainable Parameters: $\{\alpha_{i,j}^{p}\}$

Feature Interaction Search-AutoGroup

Interaction Stage:



Feature set representation:

$$g_j^p = \sum_{f_i \in s_j^p} w_i^p \, e_i$$

s_j^p: the j^{th} feature set for order-p feature interactions.
e_i: embedding for feature f_i
w_i^p: weights of embeddings in feature set s_j^p.

Interaction at a given order:

- The order-p interaction in a given set s_j^p is:

$$I_j^p = \begin{cases} (g_j^p)^p - \sum_{f_i \in s_j^p} (w_i^p e_i)^p \in R, & p \geq 2 \\ g_j^p \in R^k, & p = 1 \end{cases}$$

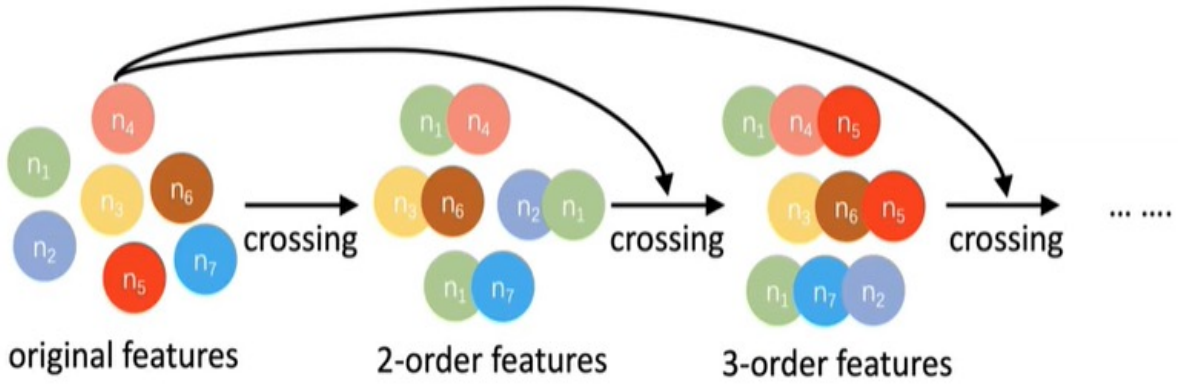
Feature Interaction Search-FIVES

The limitation of AutoGroup:

- Solve the ***efficiency-accuracy dilemma*** via feature grouping
- Ignore the ***Order-priority property***
  - ➢ The higher-order feature interactions quality can be relevant to their de-generated low-order ones

Solution of FIVES:

- Regard the original features as a **feature graph** and model the high-order feature interactions by the multilayer convolution of **GNN**, reducing the pth-order search space from $2^{C_m^p}$ to 2^{m^2}.
- Parameterize the **adjacency matrix** and make them depend on the previous layer.



FIVES: Feature Interaction via Edge Search for Large-Scale Tabular Data. KDD 2021

Feature Interaction Search-FIVES

- With an adjacency tensor A, the dedicated **graph convolutional** operator produces the node representations **layer-by-layer**. For the k-order:

$$\mathbf{n}_i^{(k)} = \mathbf{p}_i^{(k)} \odot \mathbf{n}_i^{(k-1)}$$
$$\text{where} \quad \mathbf{p}_i^{(k)} = \text{MEAN}_{j|\mathbf{A}_{i,j}^{(k)}=1}\{\mathbf{W}_j \mathbf{n}_j^{(0)}\}.$$

- The node representation at k-th layer corresponds to the generated $(k+1)$-order interactive features:

$$\begin{aligned}\mathbf{n}_i^{(k)} &= \text{MEAN}_{j|\mathbf{A}_{i,j}^{(k)}=1}\{\mathbf{W}_j \mathbf{n}_j^{(0)}\} \odot \mathbf{n}_i^{(k-1)} \\ &\approx \text{MEAN}_{(c_1,\ldots,c_k)|\mathbf{A}_{i,c_j}^{(j)}=1,j=1,\ldots,k}\{f_{c_1} \otimes \cdots \otimes f_{c_k} \otimes f_i\},\end{aligned}$$

Feature Interaction Search-FIVES

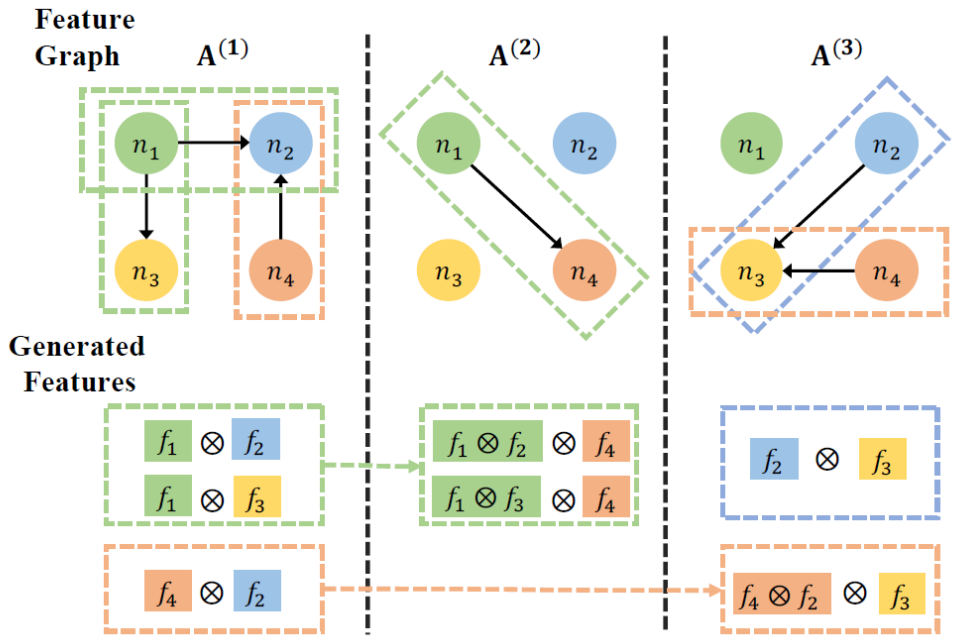
- The task of generating useful interactive features is equivalent to learning an optimal adjacency tensor A, namely **edge search**.
- The edge search task could be formulated as a **bi-level optimization problem**:

$$\begin{aligned} &\min_{A} \mathcal{L}(\mathcal{D}_{\text{val}}|A, \Theta(A)) \\ \text{s.t.} \quad &\Theta(A) = \arg\min_{\Theta} \mathcal{L}\left(\mathcal{D}_{\text{train}}|A, \Theta\right) \end{aligned} \tag{3}$$

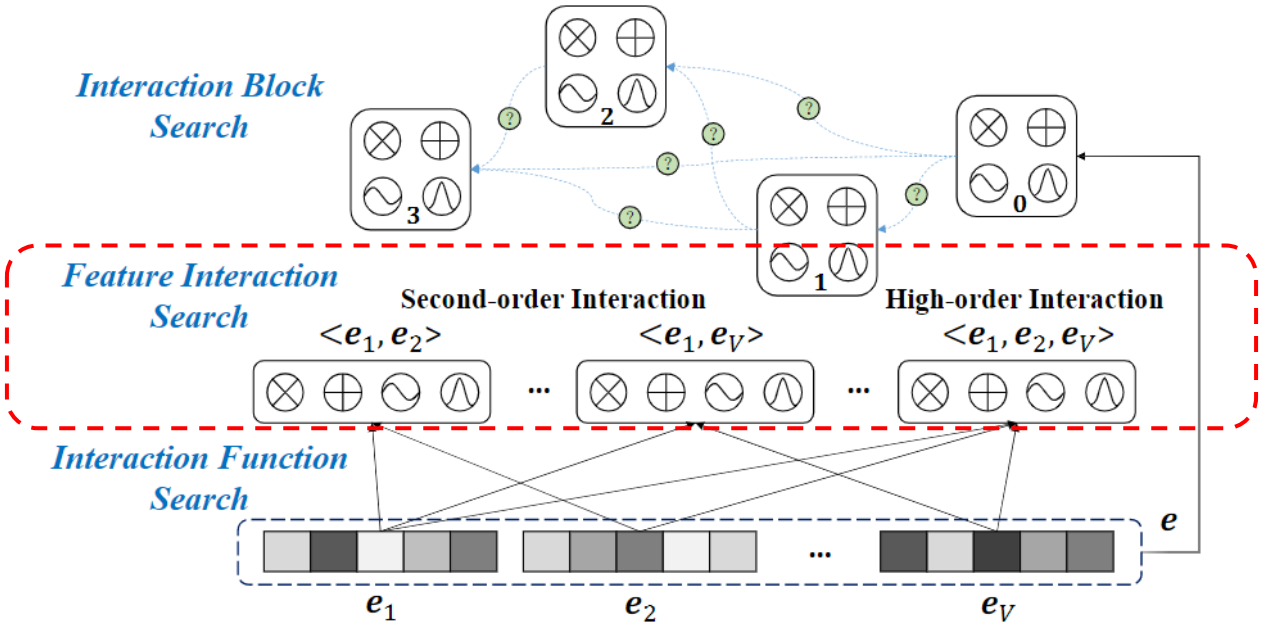
- To make the optimization more efficient, FIVES uses a soft $A^{(k)}$ for propagation at the k-th layer, while the calculation of $A^{(k)}$still depends on a binarized $A^{(k-1)}$:

$$A^{(k)} \triangleq \left(D^{(k-1)}\right)^{-1} \varphi\left(A^{(k-1)}\right) \sigma\left(H^{(k)}\right) \tag{4}$$

Degree matrix of $A^{(k-1)}$ — Binarize the soft $A^{(k-1)}$ — Interactions at k-th layer

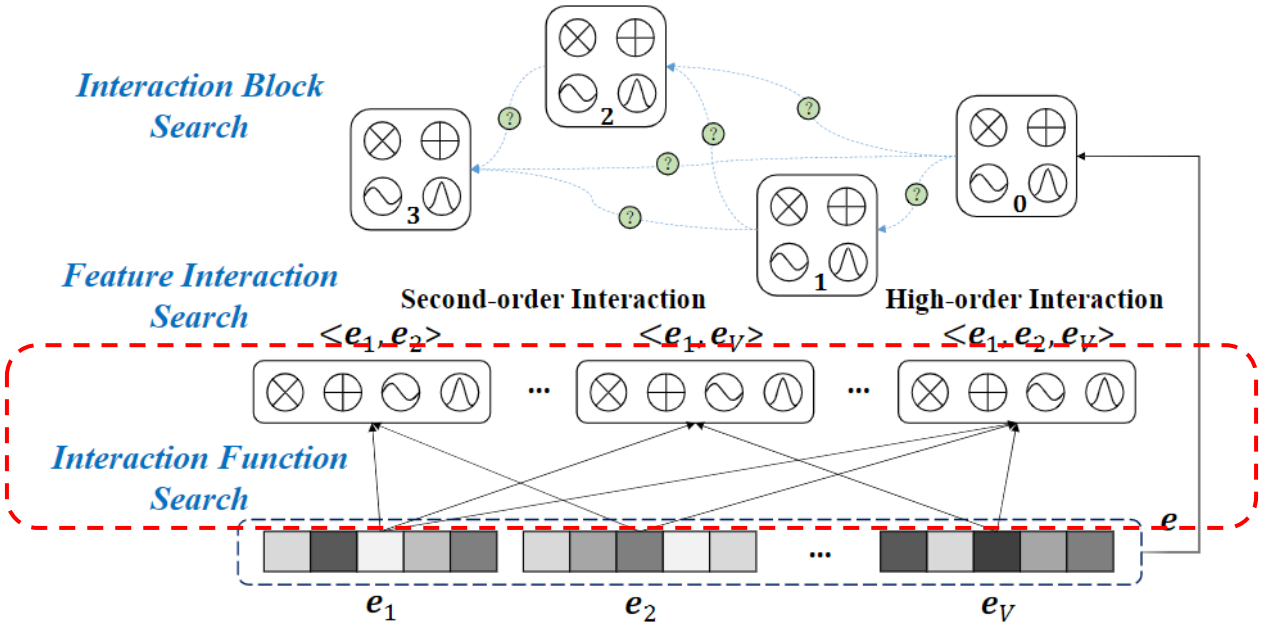


Feature Interaction Search——Summary



- Feature interaction search based methods focus on searching beneficial low-/high order interactive signals for factorization models;
- For high-order interaction search, different approaches are proposed to reduce the search space, such as feature grouping, hashing, tensor decomposition, and graph aggregation;
- Gradient-based search algorithm is dominant in this task due to the high efficiency.

Interaction Function Search



AutoML for feature interaction search:

1. Feature Interaction Search —— search beneficial feature interactions
2. **Interaction Function Search —— search suitable interaction functions**
3. Interaction Block Search —— search operations over the whole representation

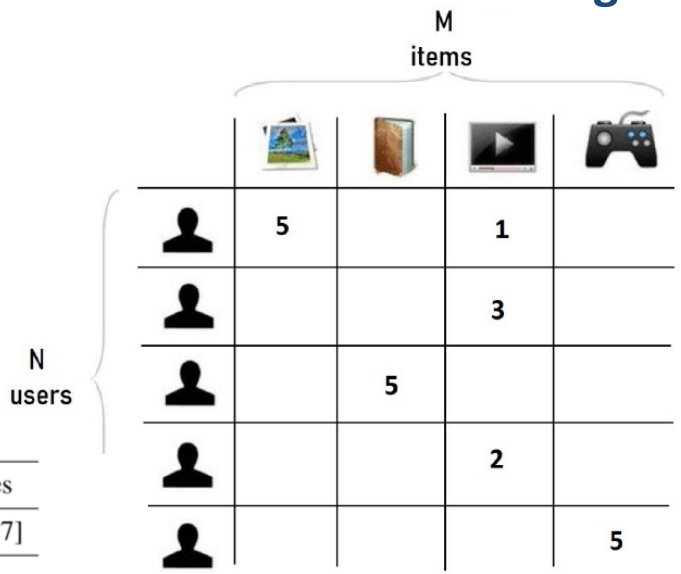
Interaction Function Search-SIF

- Generate embedding vectors for users and items
- Generate predictions by an **inner product** between embedding vectors
- Evaluate predictions by a loss function on the training data set

Interaction function:
How embedding vectors interact with each other?

	IFC	operation	space	predict time	recent examples
human-designed	$\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$	inner product	$O((m+n)k)$	$O(k)$	MF [28], FM [37]
	$\boldsymbol{u}_i - \boldsymbol{v}_j$	plus (minus)	$O((m+n)k)$	$O(k)$	CML [19]
	$\max(\boldsymbol{u}_i, \boldsymbol{v}_j)$	max, min	$O((m+n)k)$	$O(k)$	ConvMF [25]
	$\sigma([\boldsymbol{u}_i; \boldsymbol{v}_j])$	concat	$O((m+n)k)$	$O(k)$	Deep&Wide [9]
	$\sigma(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}[\boldsymbol{u}_i; \boldsymbol{v}_j])$	multi, concat	$O((m+n)k)$	$O(k^2)$	NCF [17]
	$\boldsymbol{u}_i * \boldsymbol{v}_j$	conv	$O((m+n)k)$	$O(k \log(k))$	ConvMF [25]
	$\boldsymbol{u}_i \otimes \boldsymbol{v}_j$	outer product	$O((m+n)k)$	$O(k^2)$	ConvNCF [16]
AutoML	SIF (proposed)	searched	$O((m+n)k)$	$O(k)$	——

Collaborative filtering



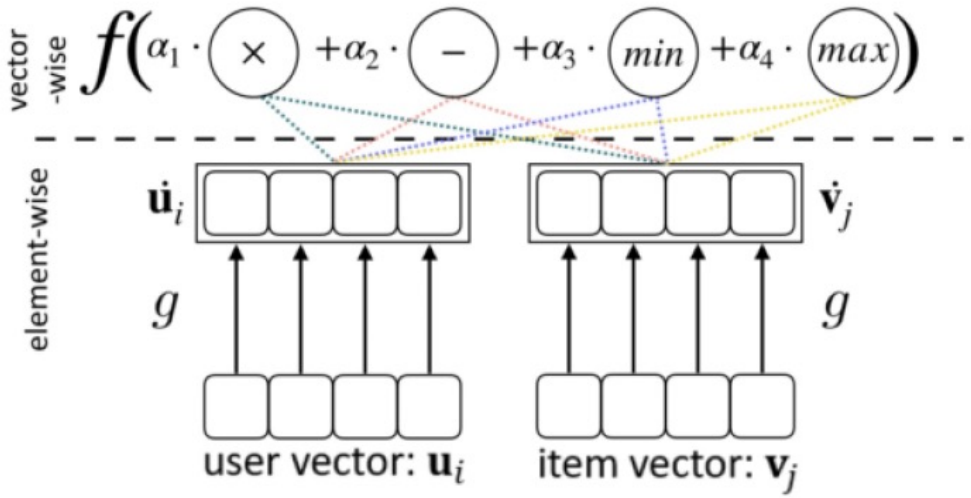
Is there an absolute best IFC? : NO, depends on tasks and datasets [1]

Efficient Neural Interaction Function Search for Collaborative Filtering. WWW 2020

Interaction Function Search-SIF

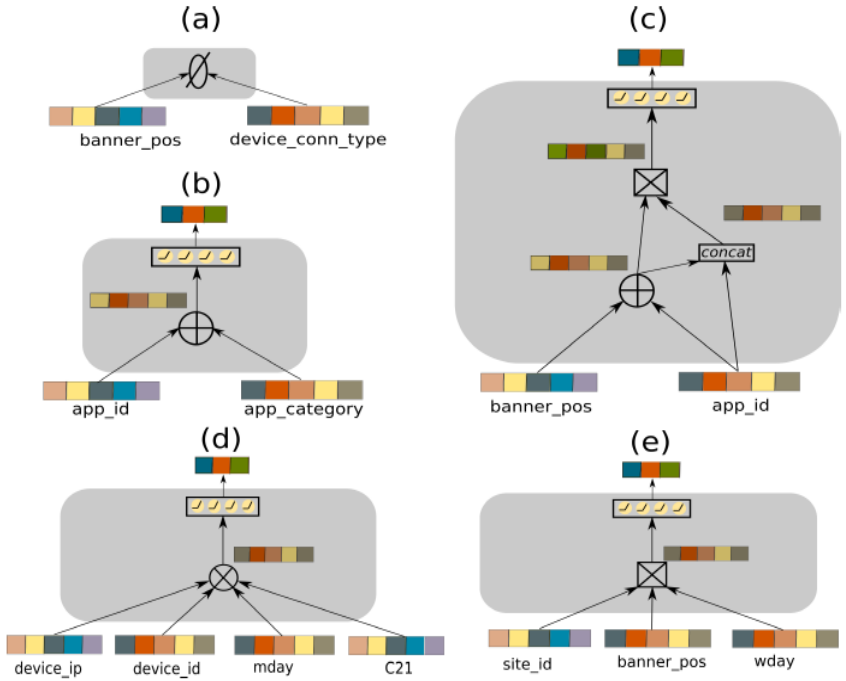
- SIF selects different interaction functions across different datasets.

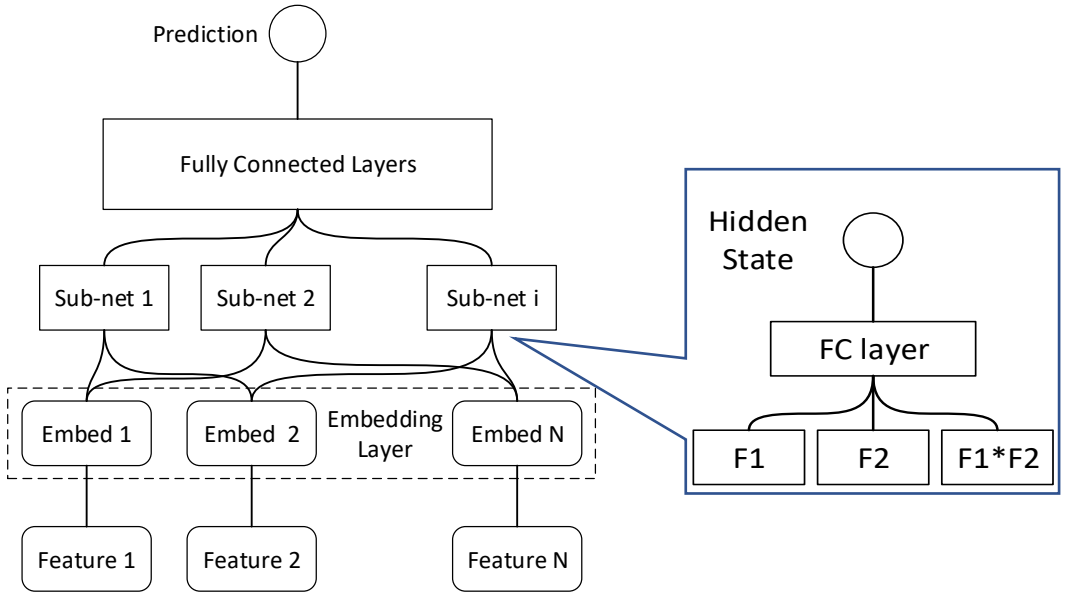
IFC	operation
$\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$	inner product
$\boldsymbol{u}_i - \boldsymbol{v}_j$	plus (minus)
$\max\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right)$	max, min
$\sigma\left([\boldsymbol{u}_i; \boldsymbol{v}_j]\right)$	concat
$\sigma\left(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}\left[\boldsymbol{u}_i; \boldsymbol{v}_j\right]\right)$	multi, concat
$\boldsymbol{u}_i * \boldsymbol{v}_j$	conv
$\boldsymbol{u}_i \otimes \boldsymbol{v}_j$	outer product



Interaction Function Search-AutoFeature

- Not all the feature interactions between each pair of fields need to be modeled.
- Not all the useful feature interactions can be modeled by the same interaction functions.



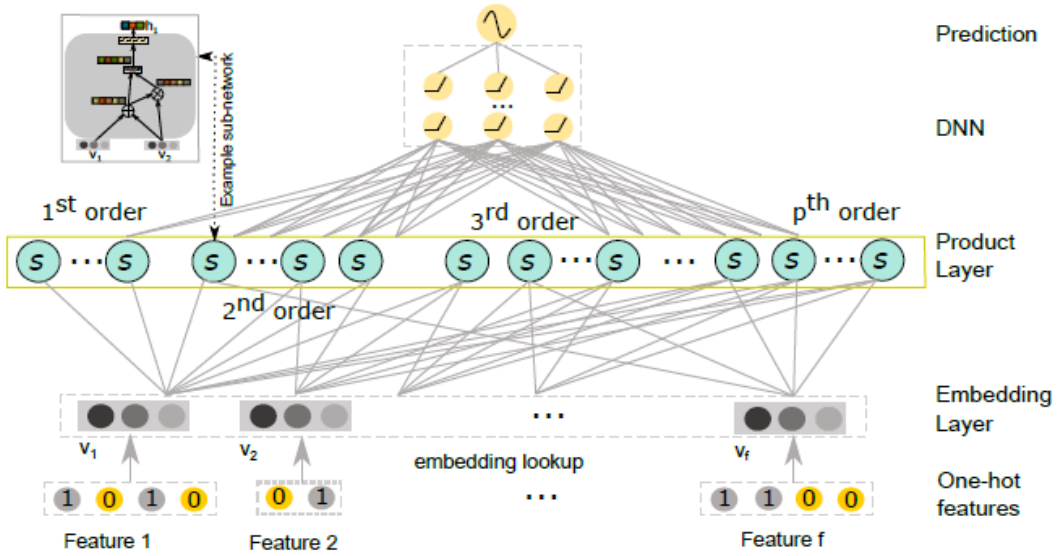


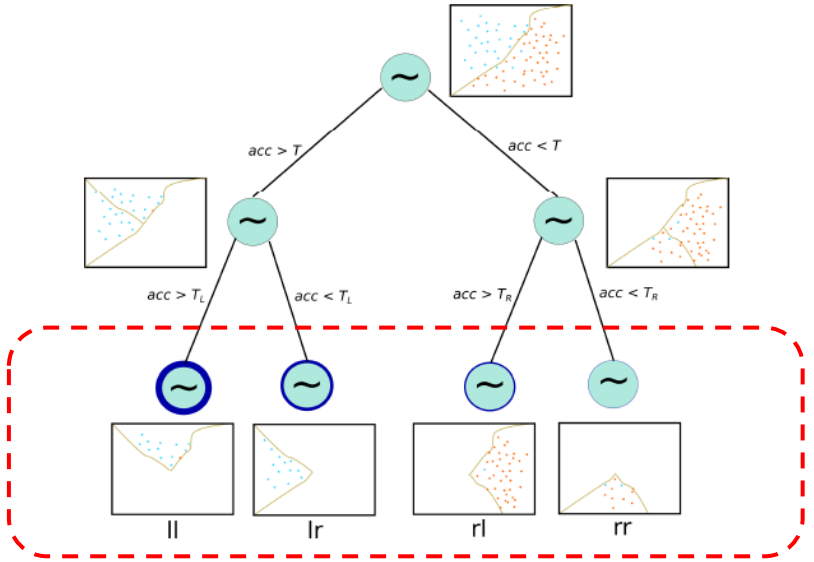
AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. CIKM 2020

Interaction Function Search-AutoFeature

AutoFeature automatically designs a different sub-net for each pair of fields.

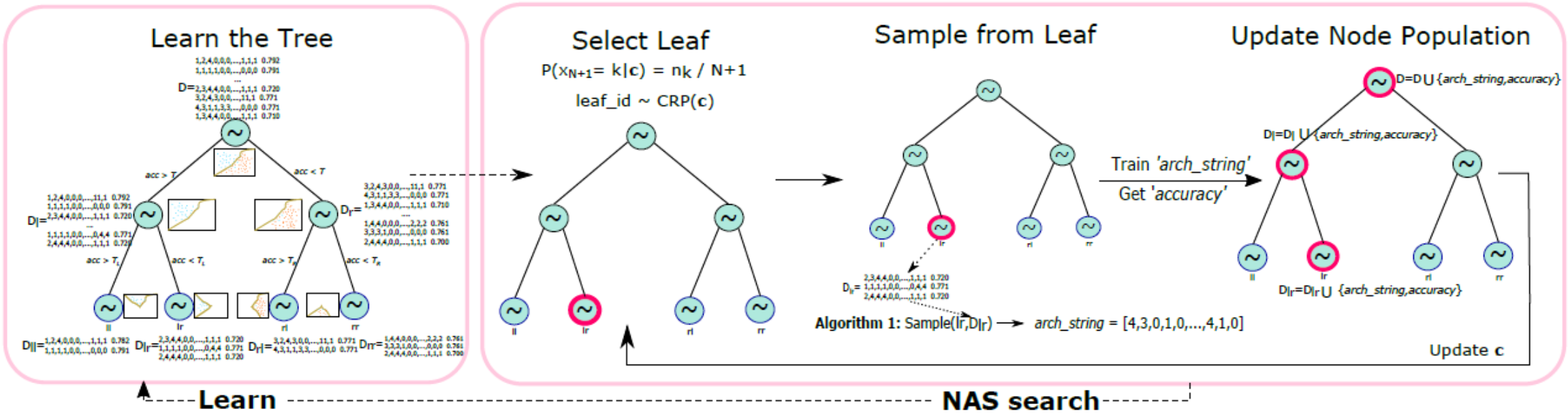
- **Train a Naïve Bayes Tree** to classify different network structures, where the tree tends to classify the **most well-performed network** into the **leftmost leaf subspace**.
- **Sample leaf nodes** from these leaf subspaces based on the Chinese Restaurant Process (CRP).





Interaction Function Search-AutoFeature

- The top two samples with the highest accuracy will be picked to perform a **crossover** operation at the midpoint of the architecture string, which is followed by q **mutations**.
- **Check** if the resulting architecture belongs to the subspace represented by the leaf node. If this is not the case then the procedure is repeated.
- The whole search procedure continues until the desired accuracy is achieved or the maximum number of steps is reached.



Interaction Function Search-AOANet

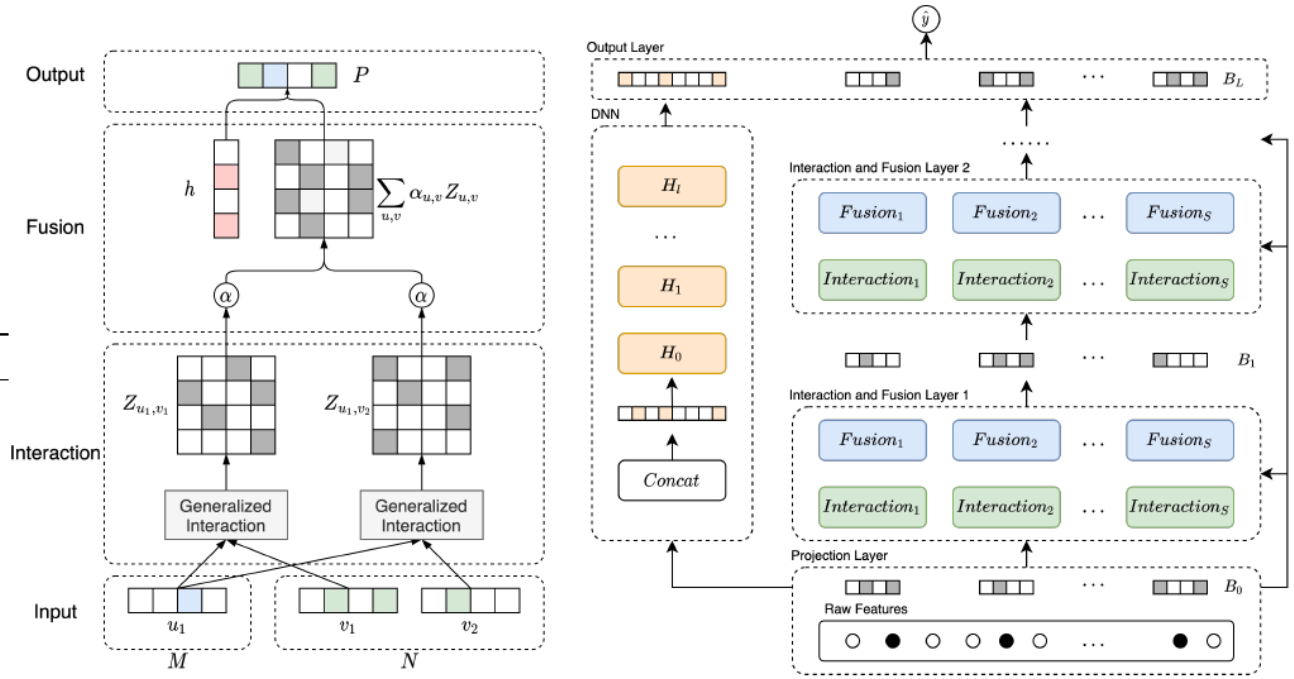
- The interaction functions of SIF and AutoFeature are artificially specified, which requires high dependence on domain knowledge.
- AOANet proposes a generalized interaction paradigm by decomposing commonly-used structures into **Projection**, **Interaction** and **Fusion** phase.
  - **Interaction**

$$\mathbf{Z}_{i,j} = (\mathbf{e}_i \otimes \mathbf{e}_j) \odot \mathbf{W},$$

  - **Fusion**

$$\mathbf{P} = \mathbf{h}^T \sum_{i,j} (\alpha_{i,j} \mathbf{Z}_{i,j}),$$

operation	related models	constraints on W	partial fusion setup
$u^T v$	FM,IPNN	$W = \text{diag}(1, \ldots, 1)$	$\sum_{i=1}^{d} Z_{i,i}$
$u \odot v$	NFM,xDeepFM	$W = \text{diag}(1, \ldots, 1)$	$[Z_{1,1}, \ldots, Z_{i,i}, \ldots, Z_{d,d}]^T$
$u \otimes v$	OPNN	$\forall i \forall j, W_{i,j} = 1$	–
$\alpha(u^T v)$	FwFM	$W = \text{diag}(\alpha, \ldots, \alpha)$	$\sum_{i=1}^{d} Z_{i,i}$
$\alpha(u \odot v)$	AFM	$W = \text{diag}(\alpha, \ldots, \alpha), \alpha = MLP(Z)$	$[Z_{1,1}, \ldots, Z_{i,i}, \ldots, Z_{d,d}]^T$
$(u \otimes v) w^D$	DCN	$\forall i \forall j \forall k, W_{i,j} = W_{i,k}$	$[\sum_i Z_{i,1}, \ldots, \sum_i Z_{i,j}, \ldots, \sum_i Z_{i,d}]^T$
$u \odot (W^D v)$	DCN-M	–	$[\sum_j Z_{1,j}, \ldots, \sum_j Z_{i,j}, \ldots, \sum_j Z_{d,j}]^T$
self-attention	AutoInt	$Z^j = (q \otimes k^j) \odot W, W = \text{diag}(1, \ldots, 1)$	$(\frac{e^{\sum_i Z_{i,i}^j}}{\sum_j e^{\sum_i Z_{i,i}^j}}) v^j$

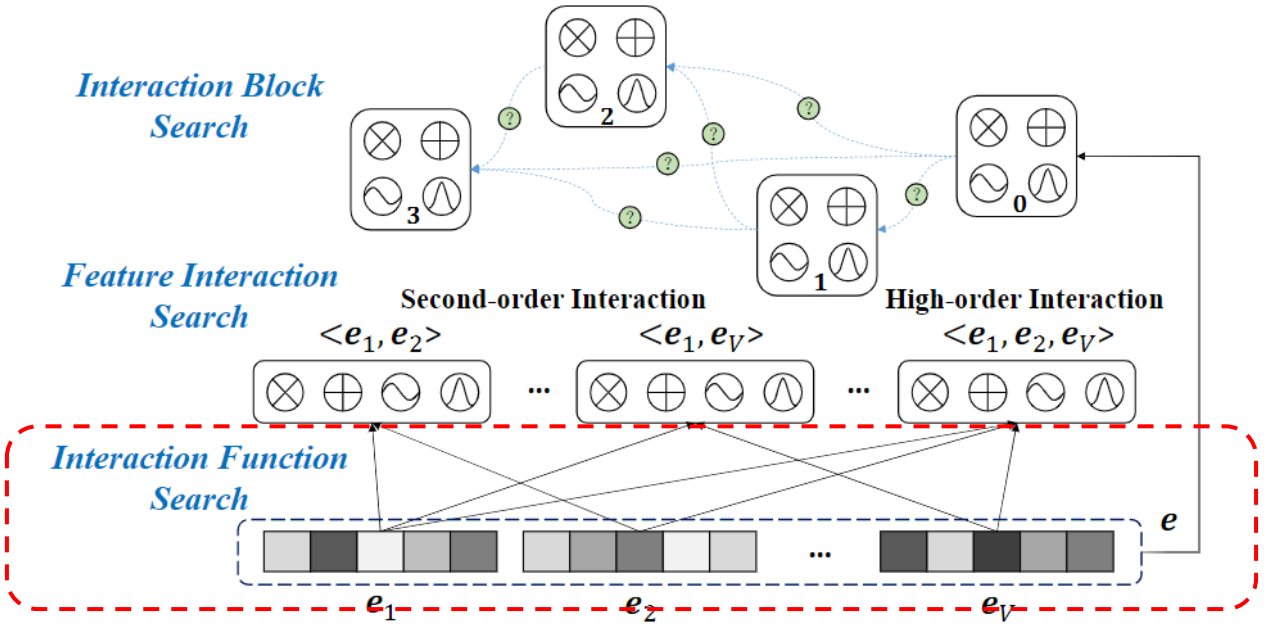


(a) Structure of interaction and fusion layer.

(b) Generalized Interaction Network(GIN)

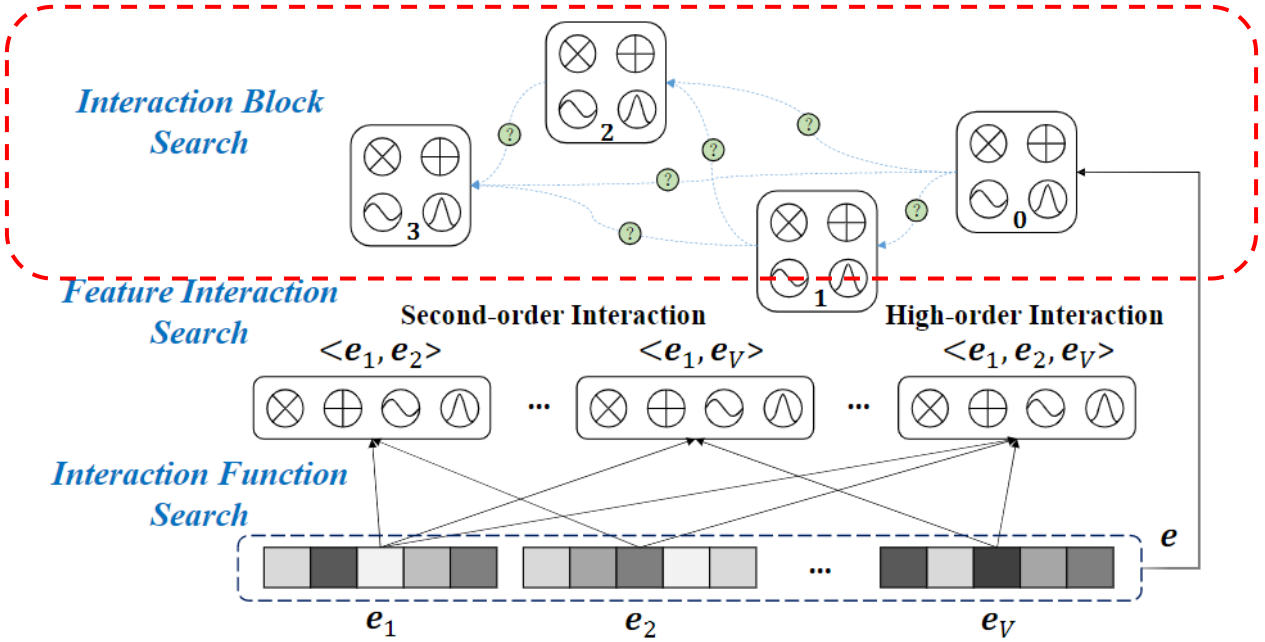
Architecture and operation adaptive network for online recommendations. KDD 2021.

Interaction Function Search——Summary



- Although searching appropriate interaction functions for different feature interactions helps to improve accuracy, the introduced cost is higher, which hinders the application in high-order scenarios;
- Generalized interaction function search (e.g., AOANet) is more efficient than searching over human-designed search space (e.g., AutoFeature), providing a promising paradigm for high-order interaction function search.

Interaction Block Search



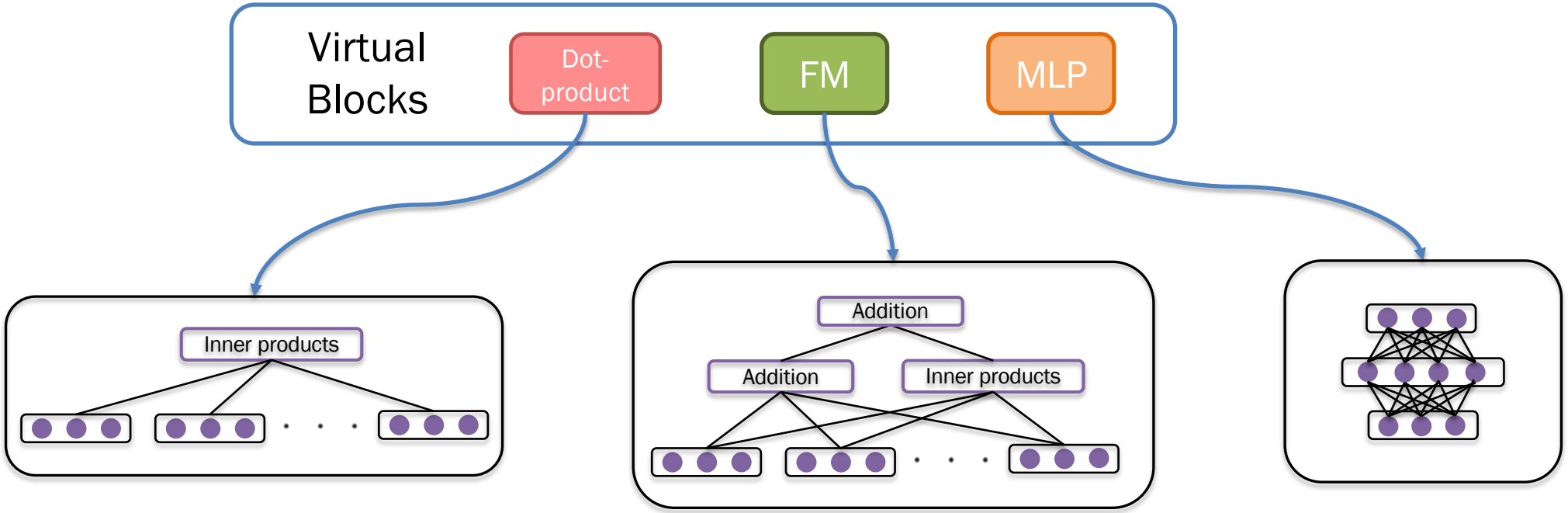
AutoML for feature interaction search:

1. Feature Interaction Search —— search beneficial feature interactions
2. Interaction Function Search —— search suitable interaction functions
3. **Interaction Block Search —— search operations over the whole representation**

Interaction Block Search-AutoCTR

Hierarchical Search Space

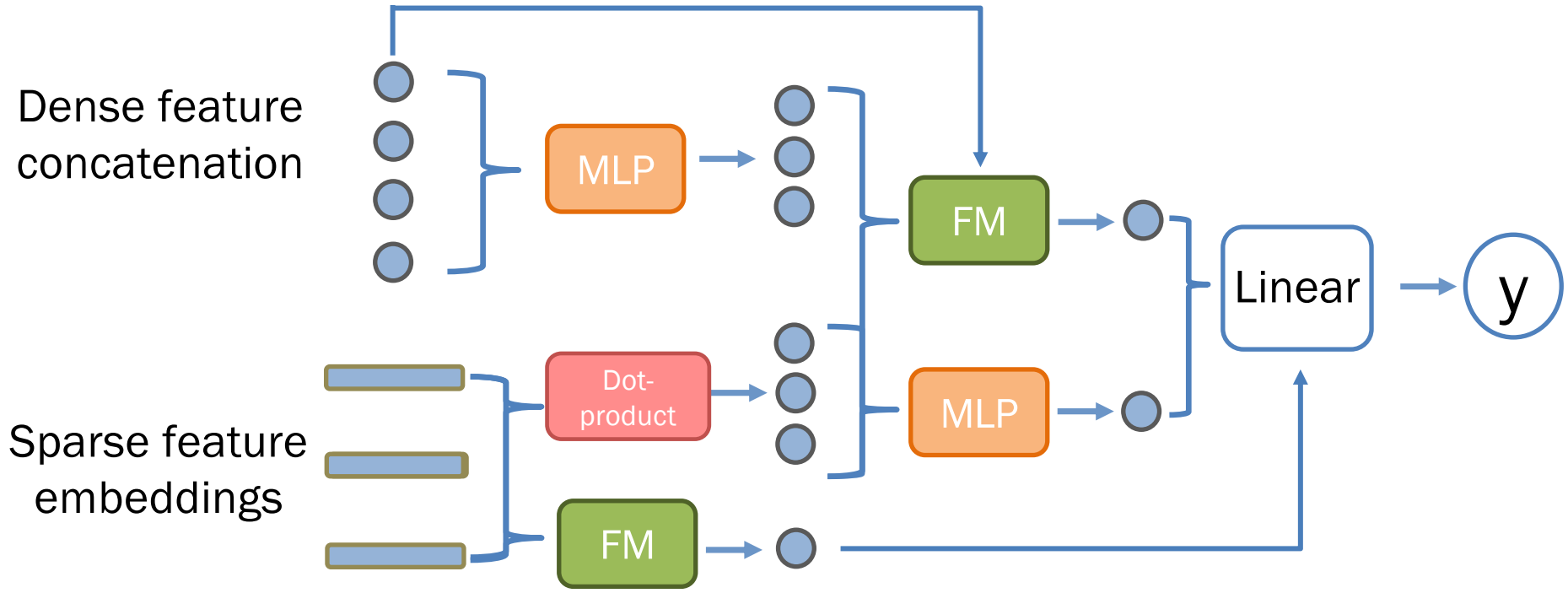
- Properties: functionality complementary, complexity aware, …
  - ✓ Inter-Block: MLP block, dot-product block, factorization-machine block, …
  - ✓ Intra-Block: Block Appendant Hyper-parameters



Interaction Block Search-AutoCTR

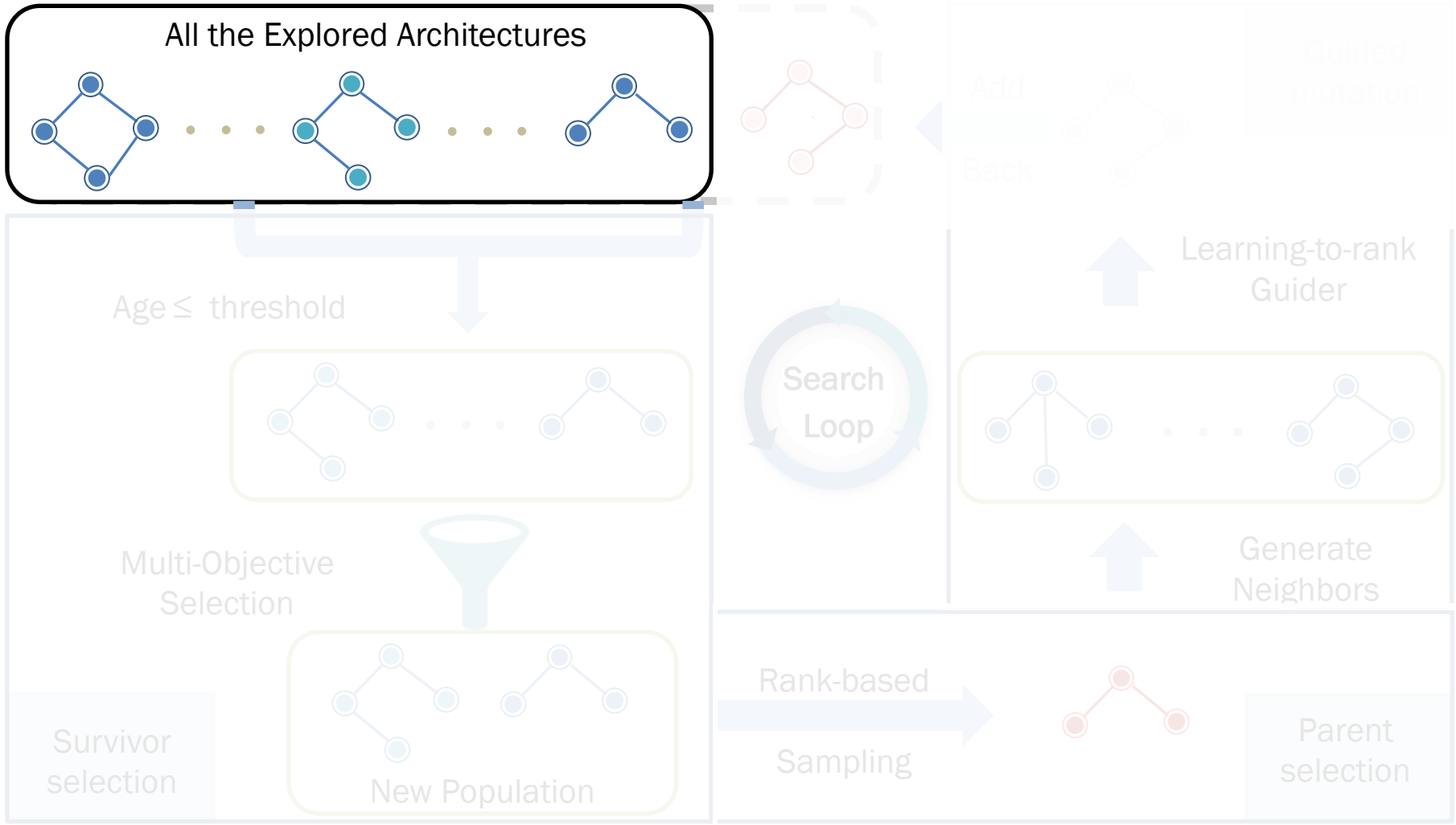
Search space construction

- DAG of virtual blocks and grouped feature embeddings
- Both block hyper-parameters (Intra-Block) and connection among blocks (Inter-Block) are to be searched



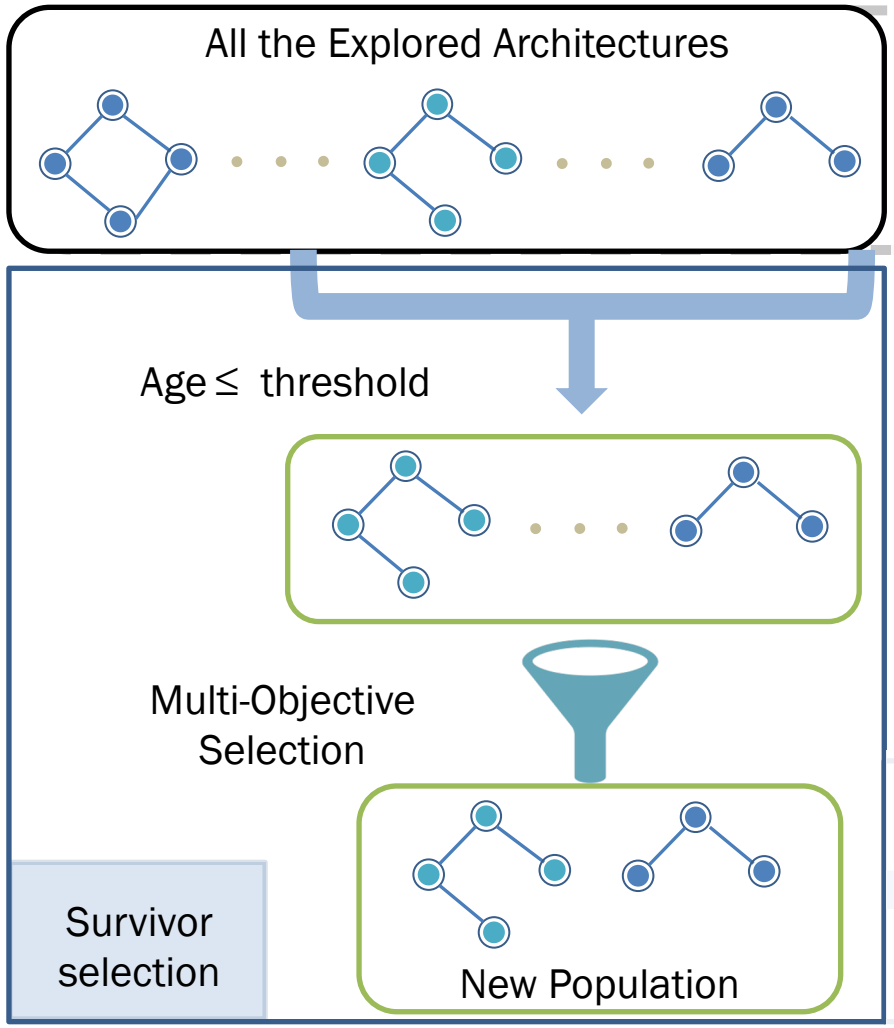
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



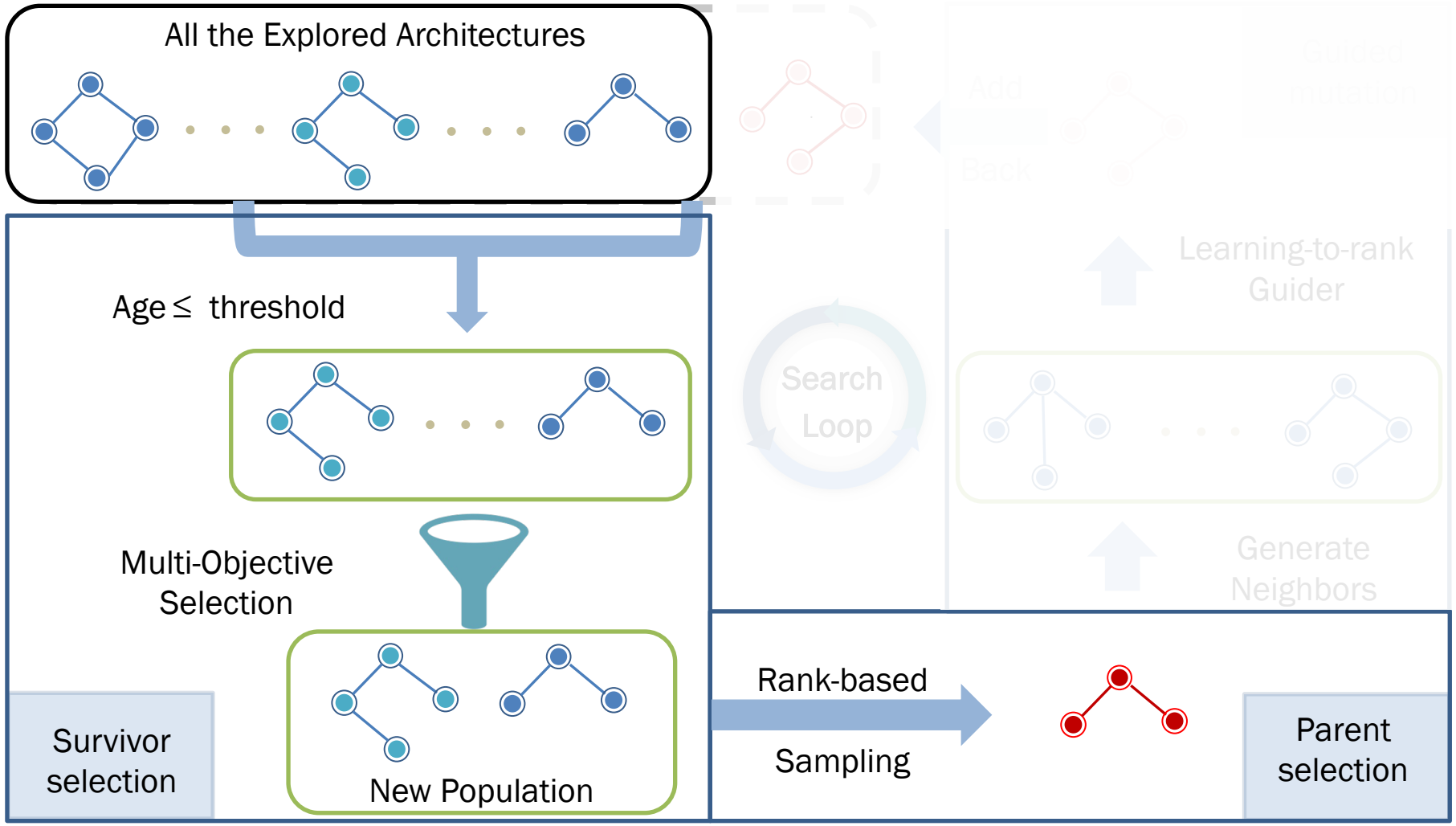
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



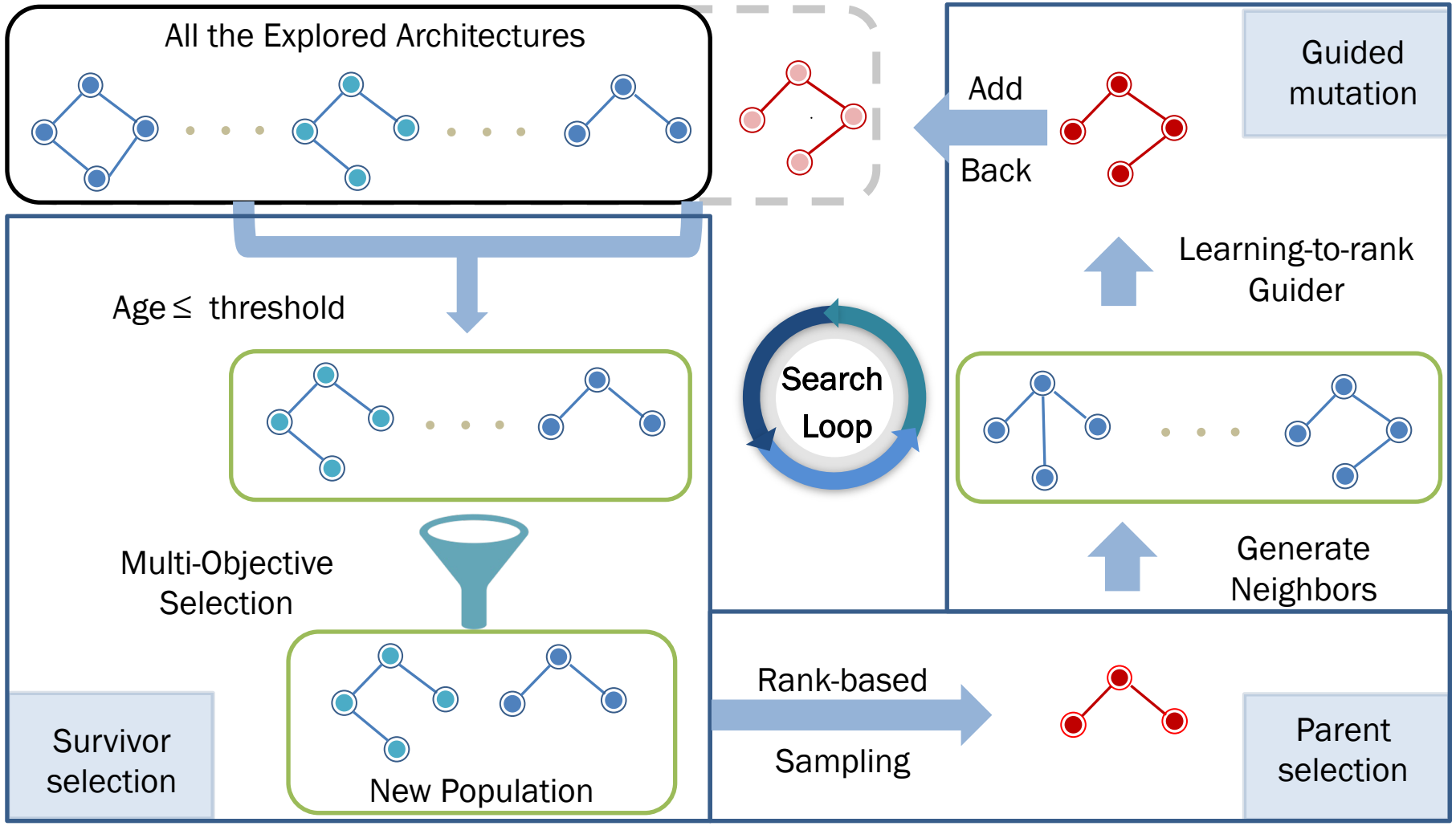
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



Interaction Block Search-AutoCTR

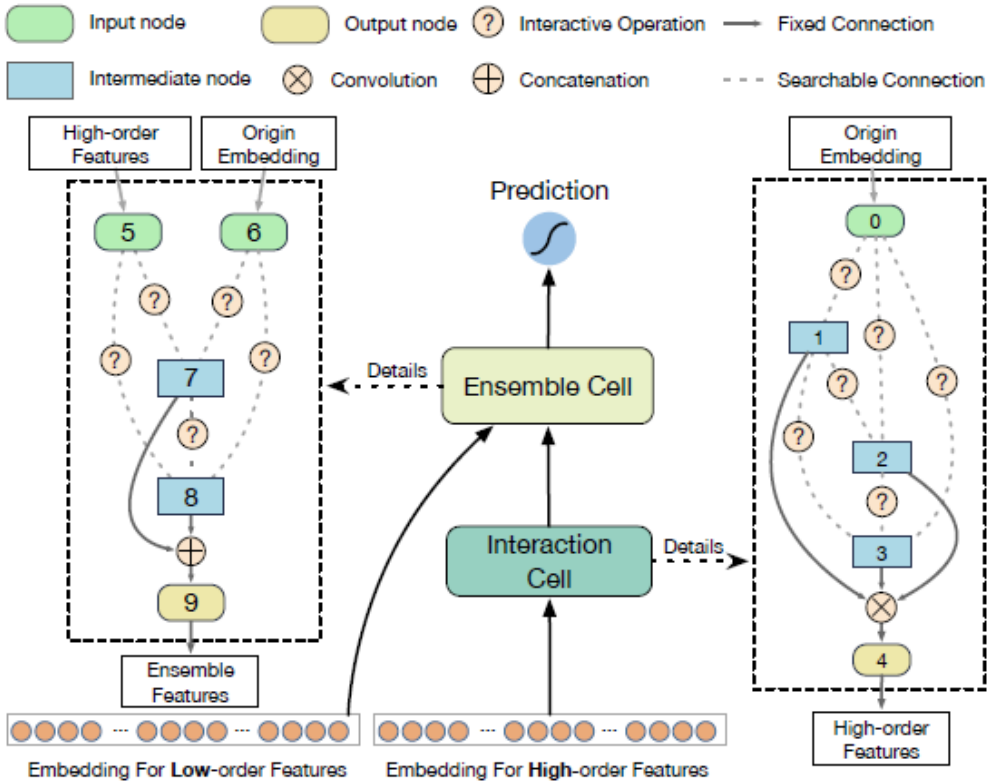
Multi-Objective Evolutionary Search Algorithm



Interaction Block Search-AutoPI

Search Space

- The interaction cell formulates the higher-order feature interactions
- The ensemble cell formulates the ensemble of lower-order and higher-order interactions



- Skip-connection[1]

$$\boldsymbol{E}' = o_{\text{skip}}(\boldsymbol{E}) = \boldsymbol{E} \in \mathbb{R}^{m \times k}$$

- SENET Layer[2]

$$\boldsymbol{E}' = \left[a'_1 \cdot \boldsymbol{x}_1, a'_2 \cdot \boldsymbol{x}_2, \cdots, a'_m \cdot \boldsymbol{x}_m\right] \in \mathbb{R}^{m \times k}$$

- Self-attention[3]

$$\boldsymbol{E}' = \left[\boldsymbol{e}_1^{\text{Res}}, \boldsymbol{e}_2^{\text{Res}}, \cdots, \boldsymbol{e}_m^{\text{Res}}\right] \in \mathbb{R}^{m \times k}$$

- FM Layer

$$\boldsymbol{p} = \{(\boldsymbol{x}_j \cdot \boldsymbol{x}_j)\}_{(i,j) \in R_x}$$
$$\text{s.t. } \boldsymbol{R}_x = \{(i,j)\}_{i \in \{1,\cdots,m\}, j \in \{1,\cdots,m\}, i<j}$$

- FC Layer

$$\boldsymbol{E}' = \boldsymbol{e} \cdot \boldsymbol{W}_{\text{FC}}$$

- 1D Conv

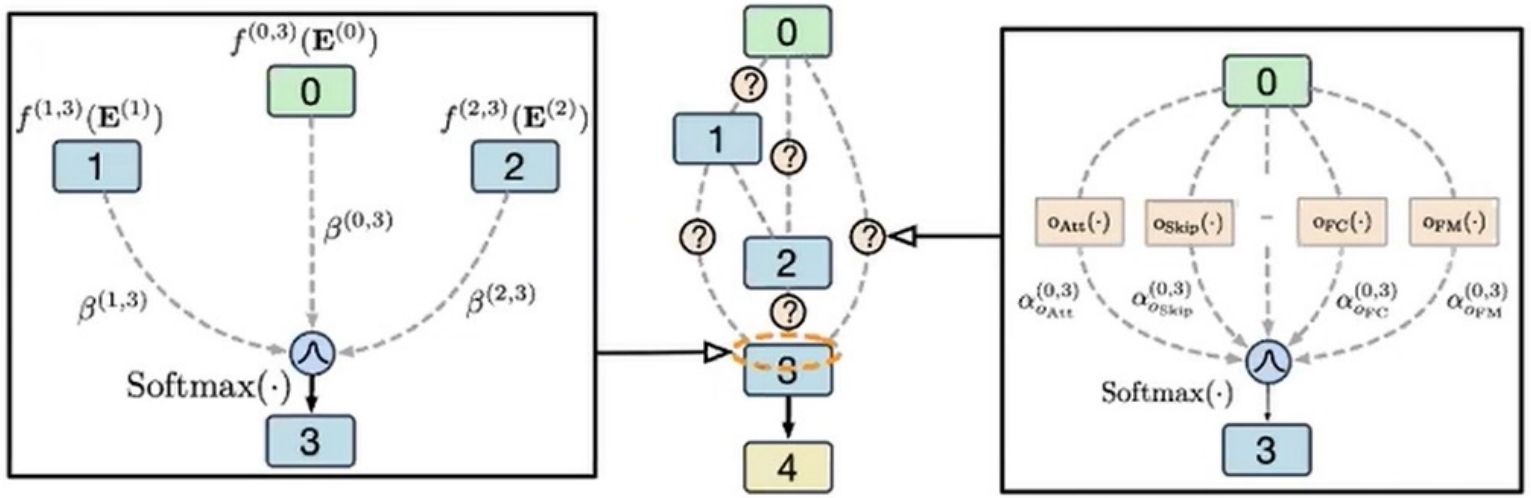
$$\left\{\boldsymbol{C}^{(i)} \in \mathbb{R}^{1 \times 1 \times m}\right\}_{i \in \{1,\cdots,m\}}$$

A General Method For Automatic Discovery of Powerful Interactions In Click-Through Rate Prediction. SIGIR 2021

Interaction Block Search-AutoPI

Search Strategy

- Continuous relaxation



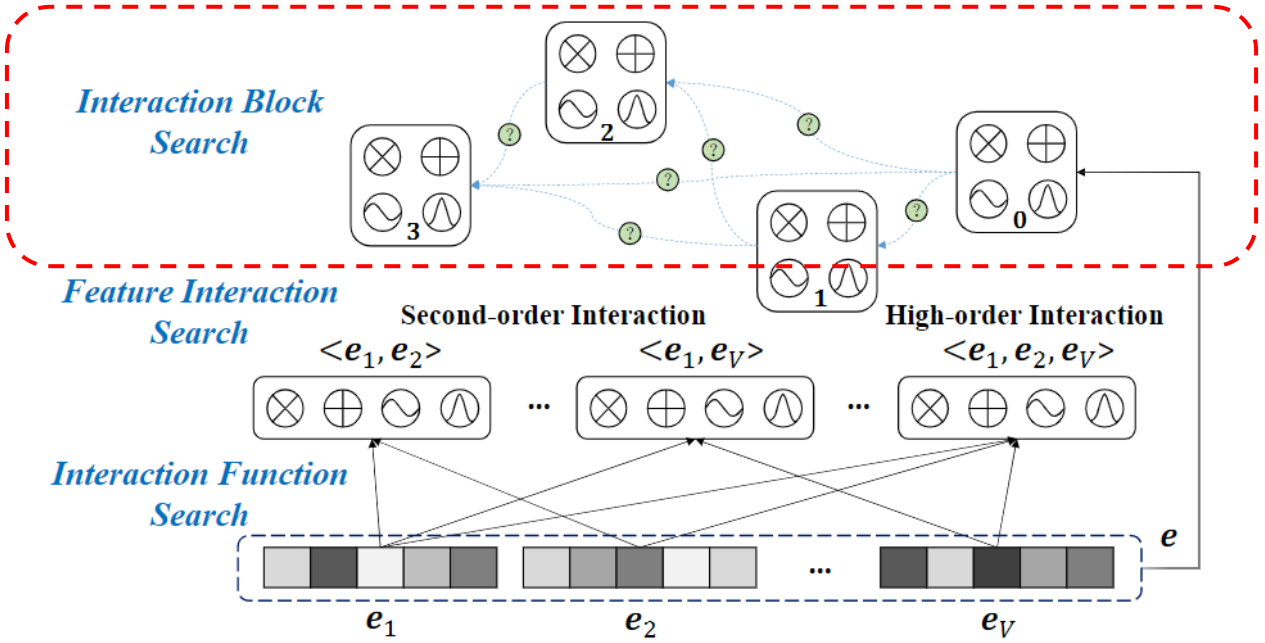
Continuous relaxation visualization

By introducing the **operator-level** and **edge-level architecture parameters** for continuous relaxation, a **differentiable objective function** will be obtained:

$$\min_{\alpha,\beta} \mathcal{L}_{val}\left(w^*(\alpha,\beta),\alpha,\beta\right)$$

$$\text{s.t. } w^*(\alpha,\beta) = \text{argmin}_w \mathcal{L}_{\text{train}}\left(w,\alpha,\beta\right)$$

Interaction Block Search——Summary



- Modeling overall high-order feature interactions over the whole feature sets implicitly can significantly shrink the search space in comparison with the explicit high-order interaction function search, making the search procedure more efficient;
- Interaction block search based methods with more abstract search space may become mainstream gradually due to its efficiency.

Summarize DRS Interaction

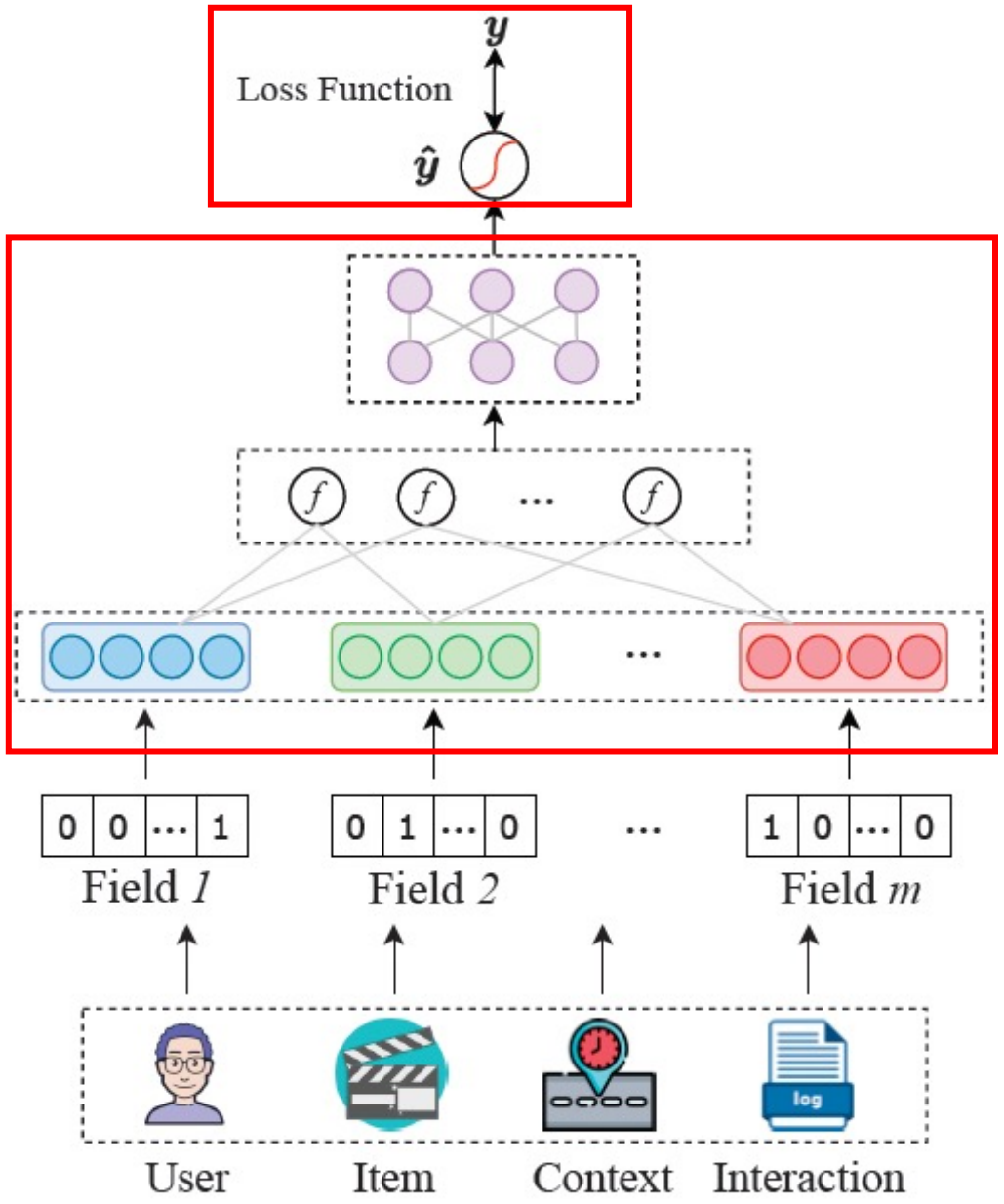
Method	Search Category	Search Space	Order	Search Algorithm
AutoFIS	Feature Interaction	$2^{C_m^p}$	second-order	Gradient
AutoGroup	Feature Interaction	2^{gm}	high-order	Gradient
PROFIT	Feature Interaction	mR	high-order	Gradient
FIVES	Feature Interaction	2^{m^2}	high-order	Gradient
L_0-SIGN	Feature Interaction	2^{m^2}	second-order	Gradient
BP-FIS	Feature Interaction	$u2^{C_m^p}$	second-order	Bayesian
SIF	Interaction Function	C_c^n	second-order	Gradient
AutoFeature	Interaction Function	$c^n nC_m^p$	second-order	Evolutionary
AOANet	Interaction Function	$2^{\lvert(i,j)\ pairs\rvert}$	high-order	Gradient
AutoCTR	Interaction Block	$c^{C_n^2}$	high-order	Evolutionary
AutoPI	Interaction Block	$c^{C_n^2}$	high-order	Gradient

* m is the number of feature fields, p is the order, g is the number of pre-defined groups, n is the number of pre-defined bolcks, c is the number of candidate interaction functions.

Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

Background



- Model training:

  Searching for architectures related to model training

- Comprehensive search:

  Searching for several parts of DRS

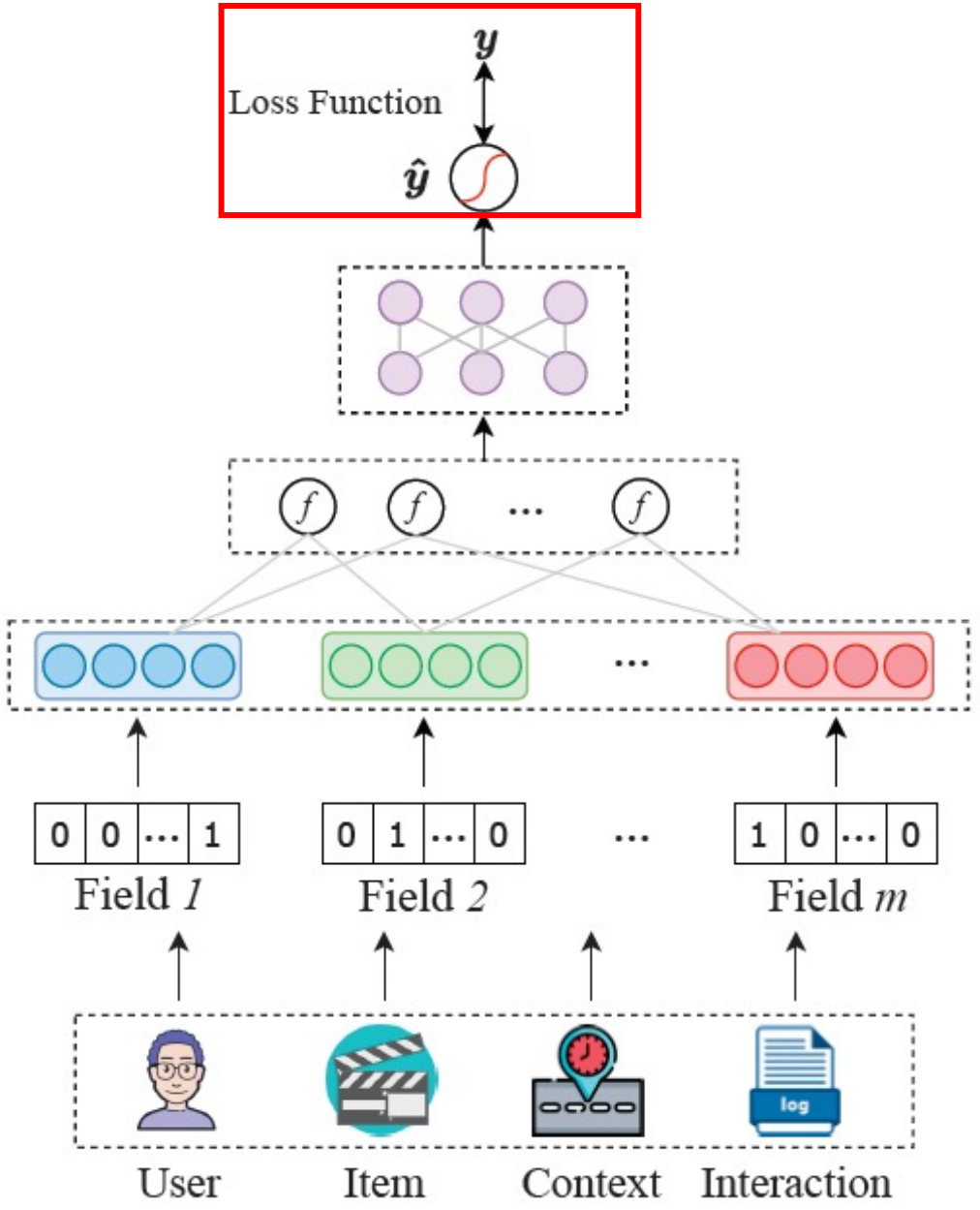
Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

AutoLoss

- Motivation:
  - Predefined and fixed loss function
  - Exhaustively or manually searched fused loss

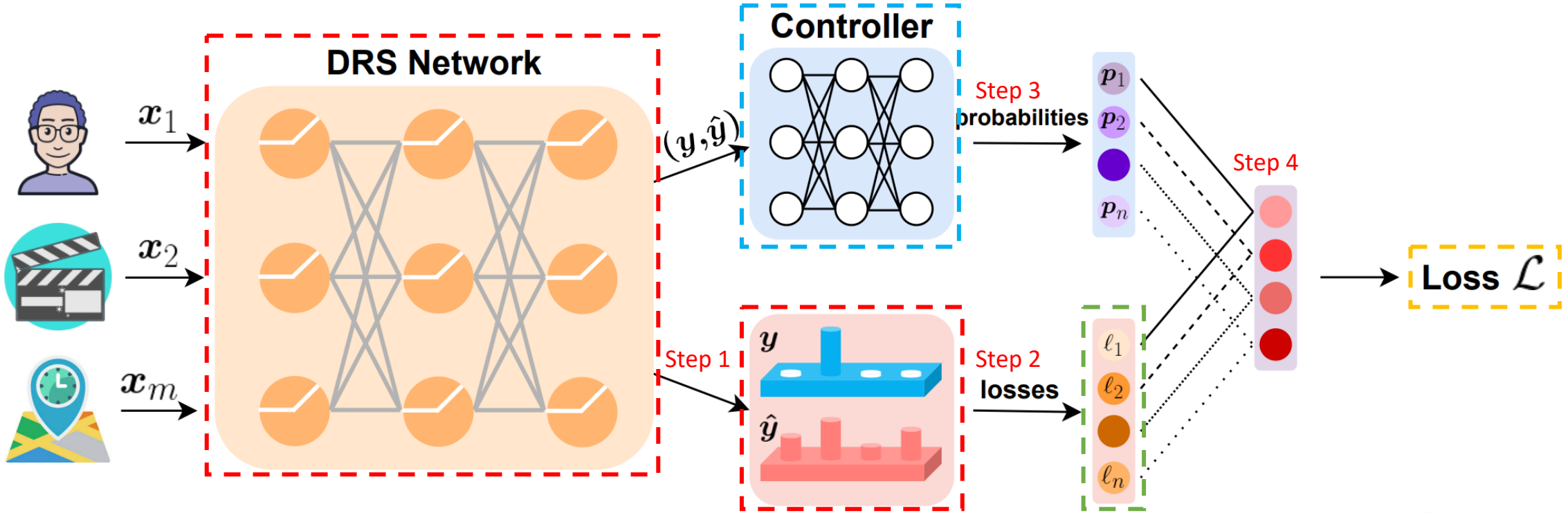
- Target:
  - Searching for loss function
  - Considering convergence behavior



AutoLoss: Automated Loss Function Search in Recommendations, KDD 21

AutoLoss – Forward-propagation

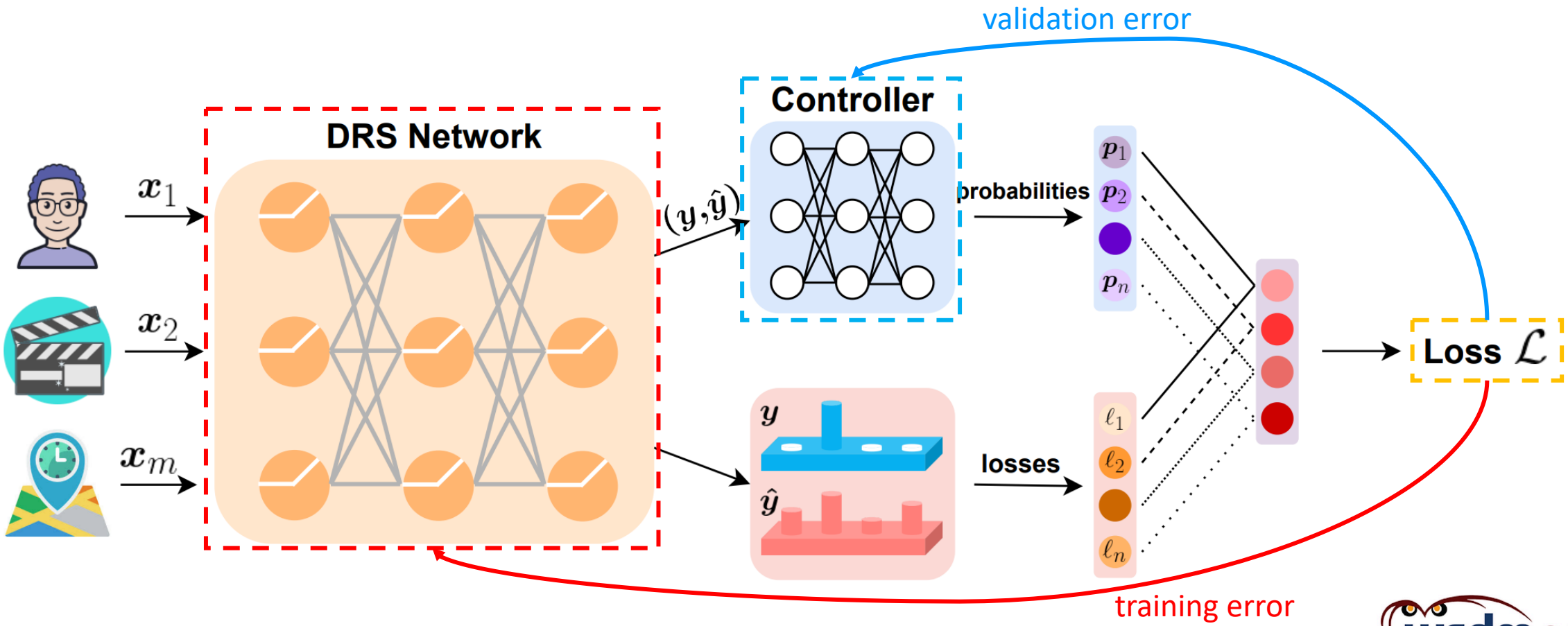
- Step 1: the DRS makes predictions
- Step 2: calculating candidate losses
- Step 3: the controller generates weights(probabilities) according to predictions
- Step 4: calculating the overall Loss (Weighted sum)



AutoLoss – Backward-propagation

- DRS network: updated based on training error

- Controller: updated based on validation error



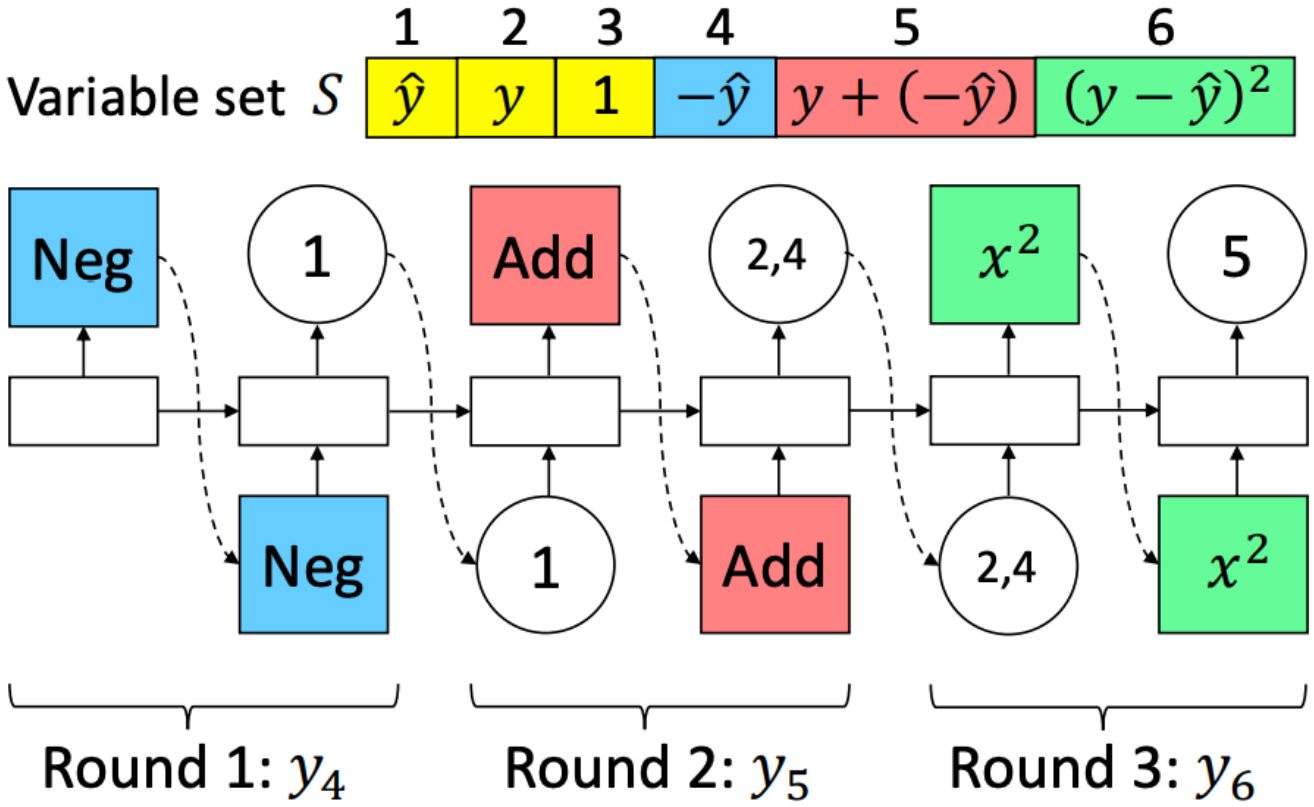
AutoLossGen

- Motivation:
  - Handcrafted loss -> expertise & efforts
  - Loss combinations -> all candidates are not suitable

- Target:
  - Generate loss functions based on basic mathematical operations

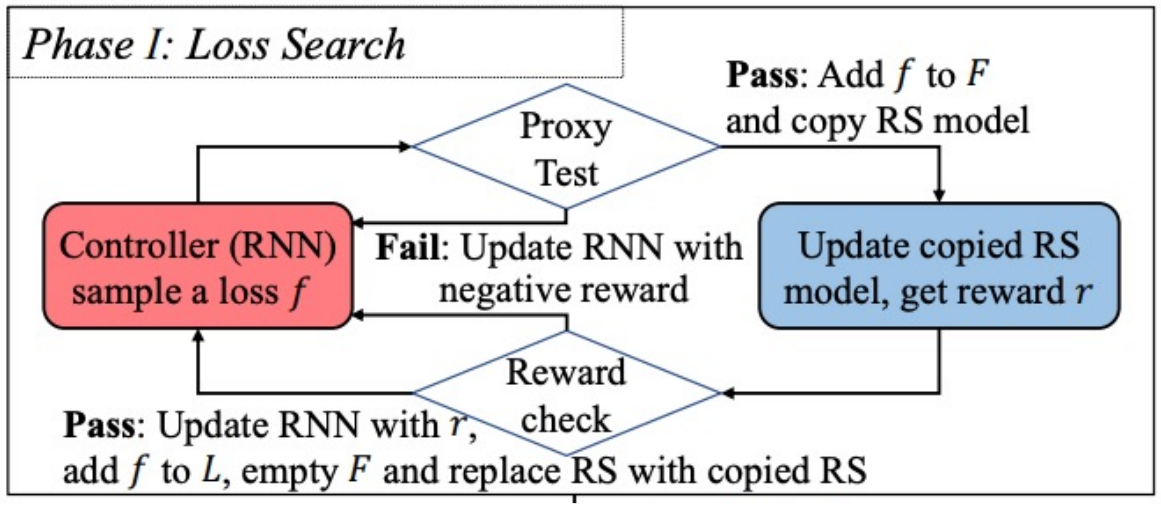
AutoLossGen

- Search space: basic operations
- Loss function generation: operation + position



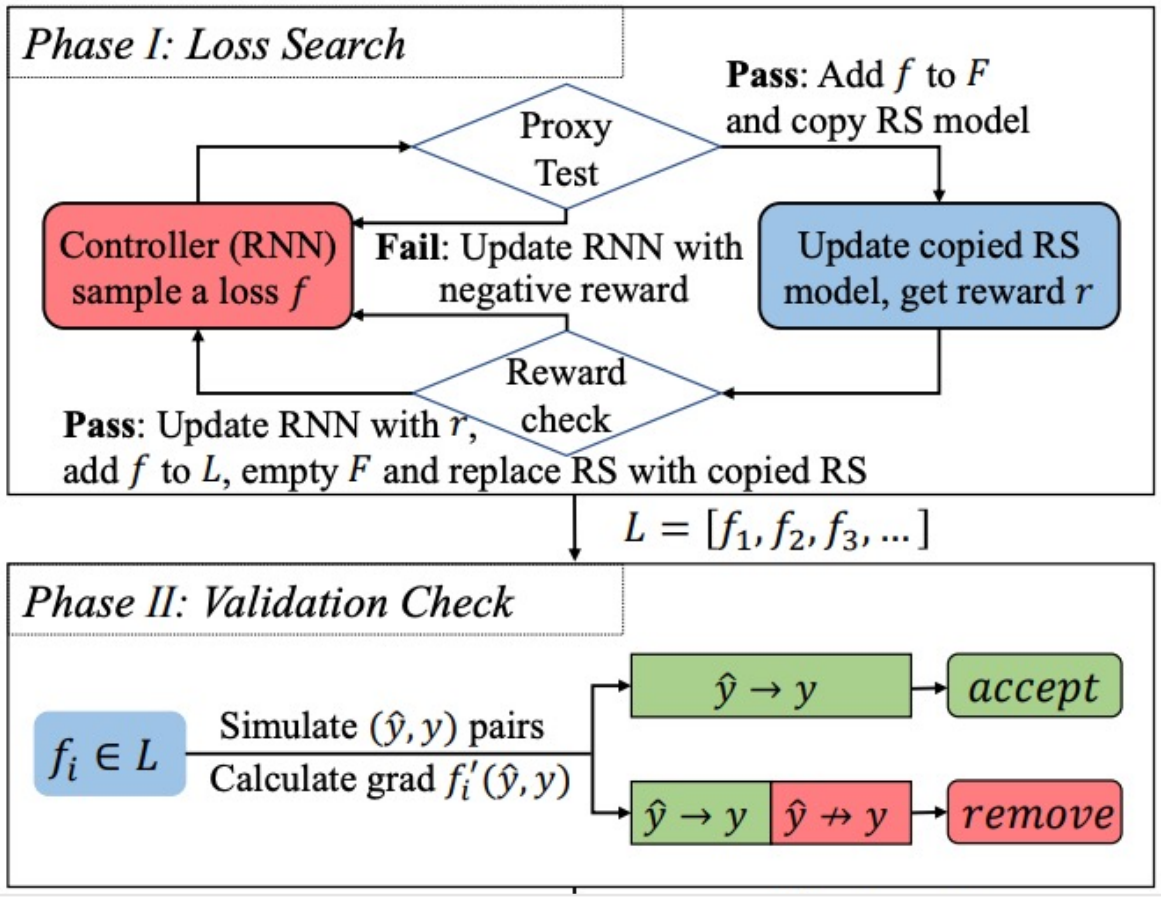
Operator	Expression	Arity
Add	$x+y$	2
Multi	$x \cdot y$	2
Max	$\max(x, y)$	2
Min	$\min(x, y)$	2
Neg	$-x$	1
Identical	x	1
Log	$\text{sign}(x) \cdot \log(\lvert x \rvert + \xi)$	1
Square	x^2	1
Reciprocal	$\text{sign}(x)/(\lvert x \rvert + \xi)$	1

AutoLossGen



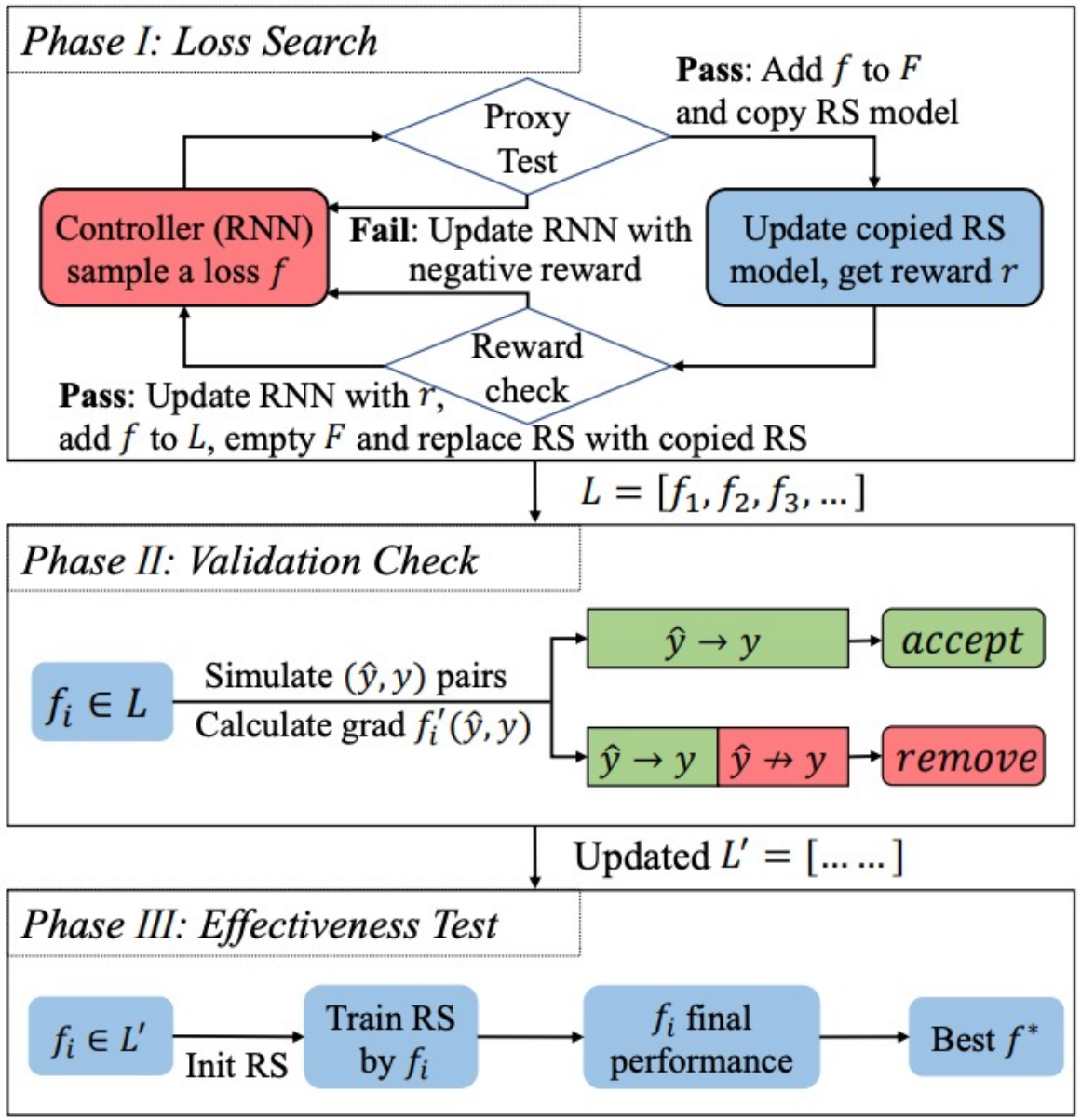
- Phase I (Search by RL)
  - Step 1: Loss function generation
  - Step 2: Check the formula
  - Step 3: One-shot evaluation (Reward)
  - Step 4: Backward

AutoLossGen



- Phase I (Search by RL)
- Phase II
  - Check the gradient

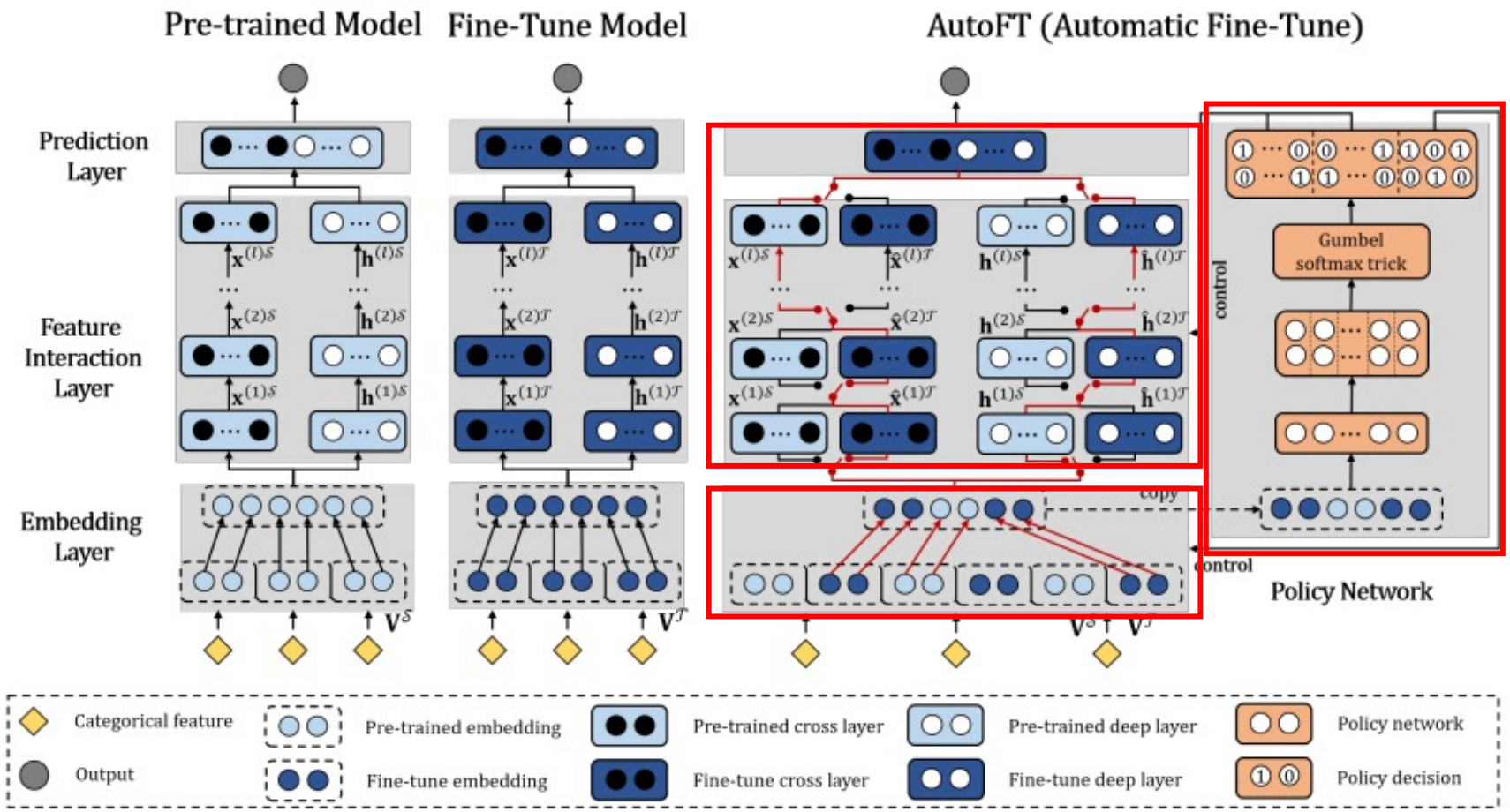
AutoLossGen



- Phase I (Search by RL)
- Phase II
  - Check the gradient
- Phase III
  - Train RS to converge

AutoFT

- Target:
  transfer learning for DRS
- Search space:
  - Field-wise transfer
  - Layer-wise transfer
- Strategy: gradient



Summarize DRS Model Training

Method	Search Space	Search Strategy
AutoLoss	Optimization: Loss Function	Gradient
AutoLossGen	Optimization: Loss Function	RL
AutoFT	Parameter Tuning: Fine-Tune or Not	Gradient

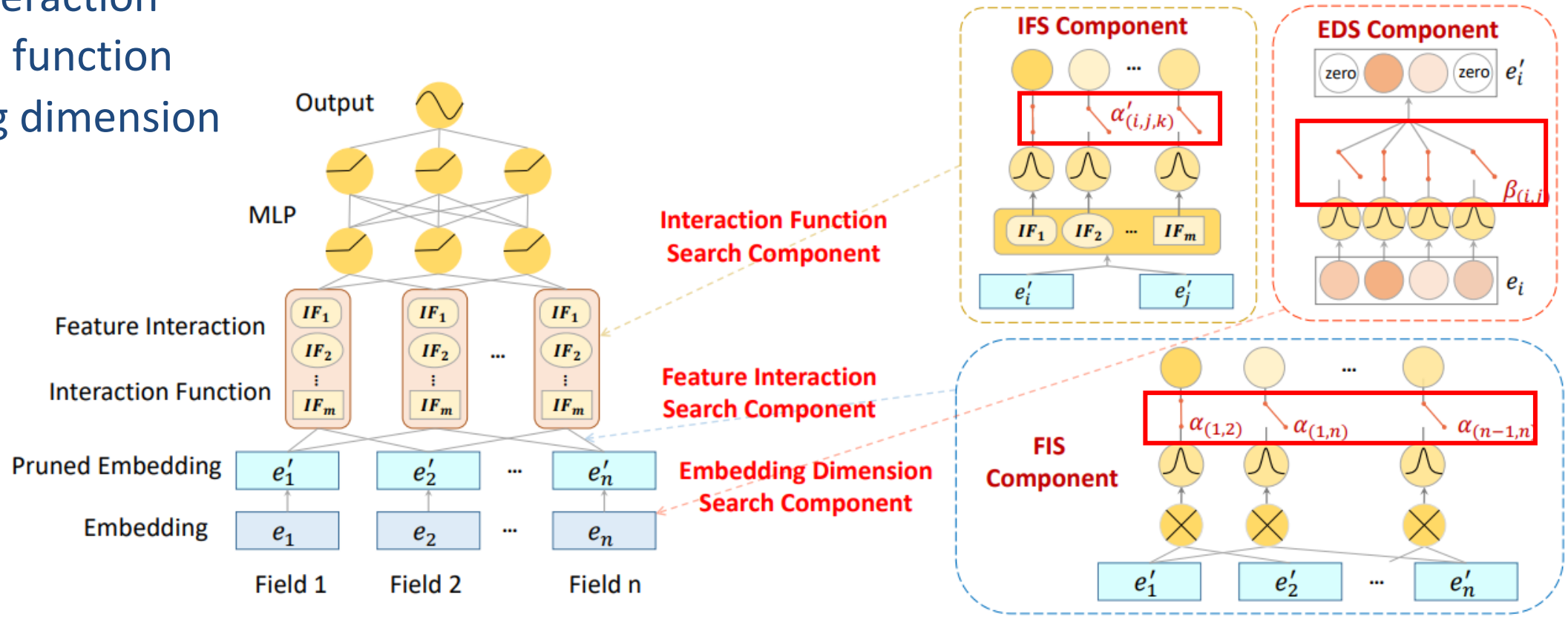
- Loss-based optimization methods facilitate recommendation model training via searching optimal loss function automatically, bringing significant results;
- From loss function to transfer learning, researchers gradually realize more flexible and efficient methodologies to facilitate model training.

Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

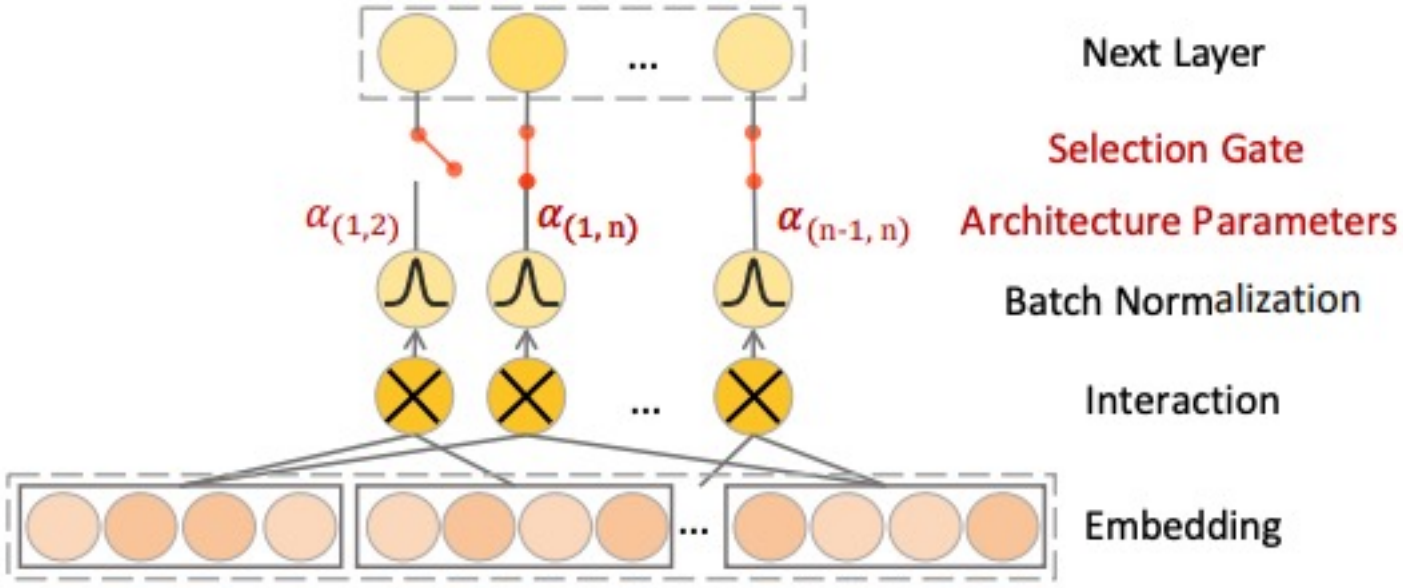
AIM

- Search space:
  - Feature interaction
  - Interaction function
  - Embedding dimension
- Strategy:
  - Gradient



AIM

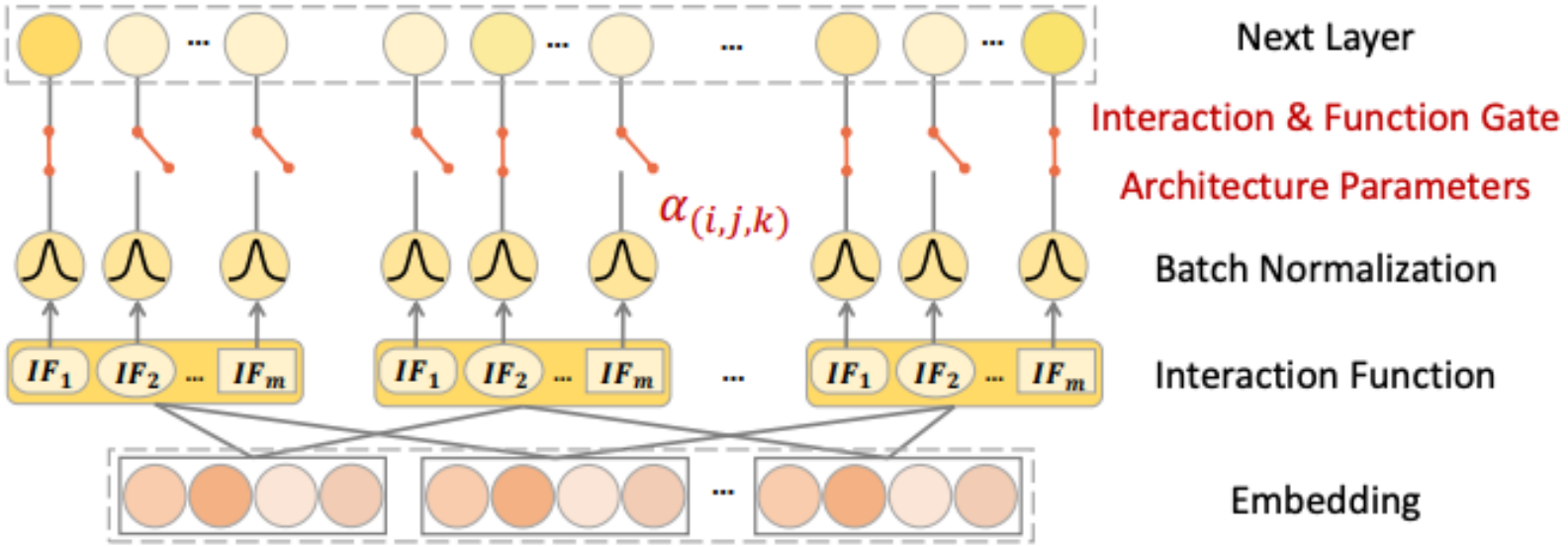
- Feature interaction search
  - Progressively enlarge feature interactions
  - For p th order: search for interaction of $p-1$ th order and 1 st



AIM

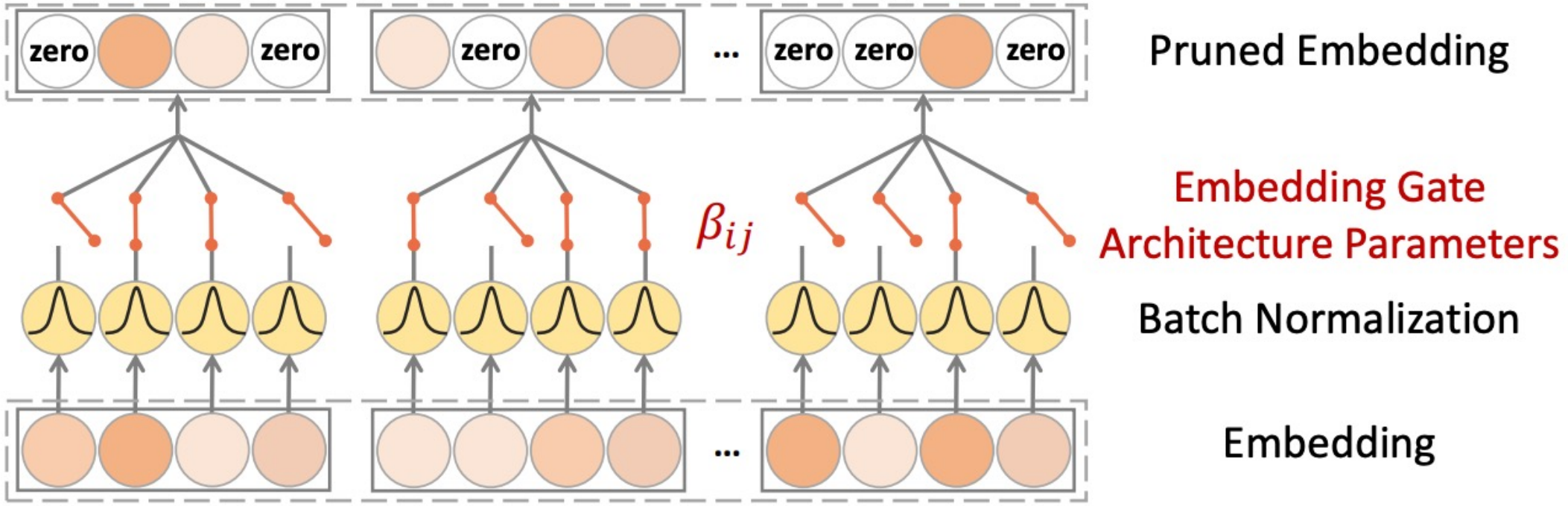
- Interaction function search
  - Consider function-wise embeddings

$$\sum_{i=1}^{n}\sum_{j>i}^{n}\sum_{k=1}^{m}\alpha'_{(i,j,k)}f_k(\boldsymbol{e}_{ik},\boldsymbol{e}_{jk})$$



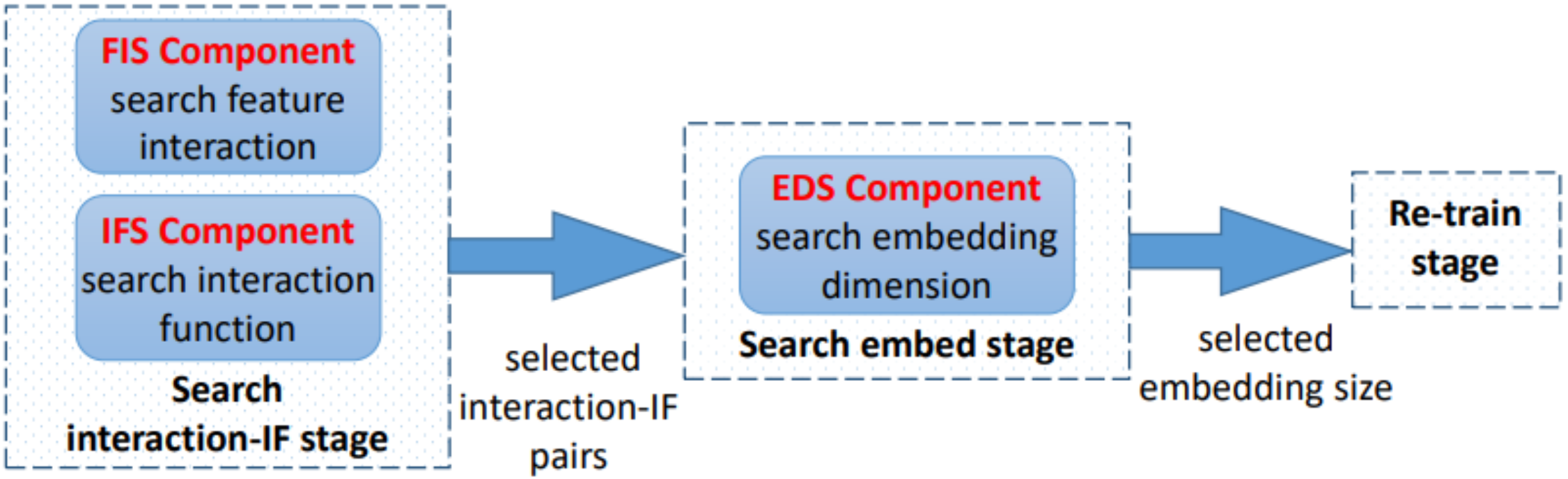
AIM

- Embedding dimension search
  - Consider the position information during the search
  - Obtain embeddings dimensions from the pruned embeddings for retraining

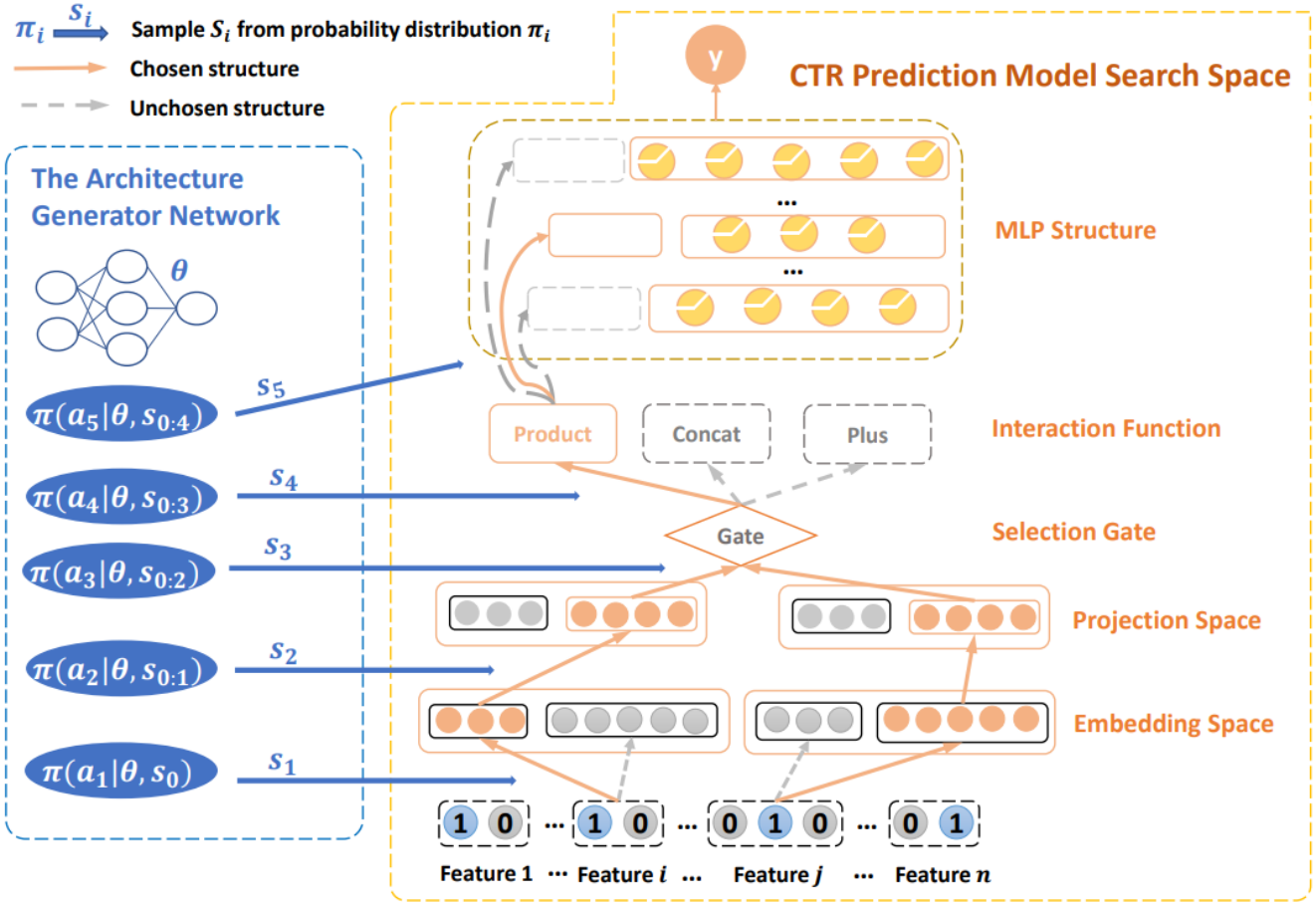


AIM

- Step 1: Search for feature interaction and feature interaction function
- Step 2: Search for embedding dimensions
- Step 3: Construct DRS and retrain



AutoIAS



- Search space:
  - Embedding size
  - Projection size
  - Feature interaction
  - Interaction function
  - MLP
- Strategy:
  - Knowledge distillation
  - Reinforcement learning

Autoias: Automatic integrated architecture searcher for click-trough rate prediction, CIKM 2021

AutoIAS

- Embedding size (S1)

$$\{d_1, d_2, ..., d_n\}$$

- Projection size (S2)
  - Unified embedding for feature interaction

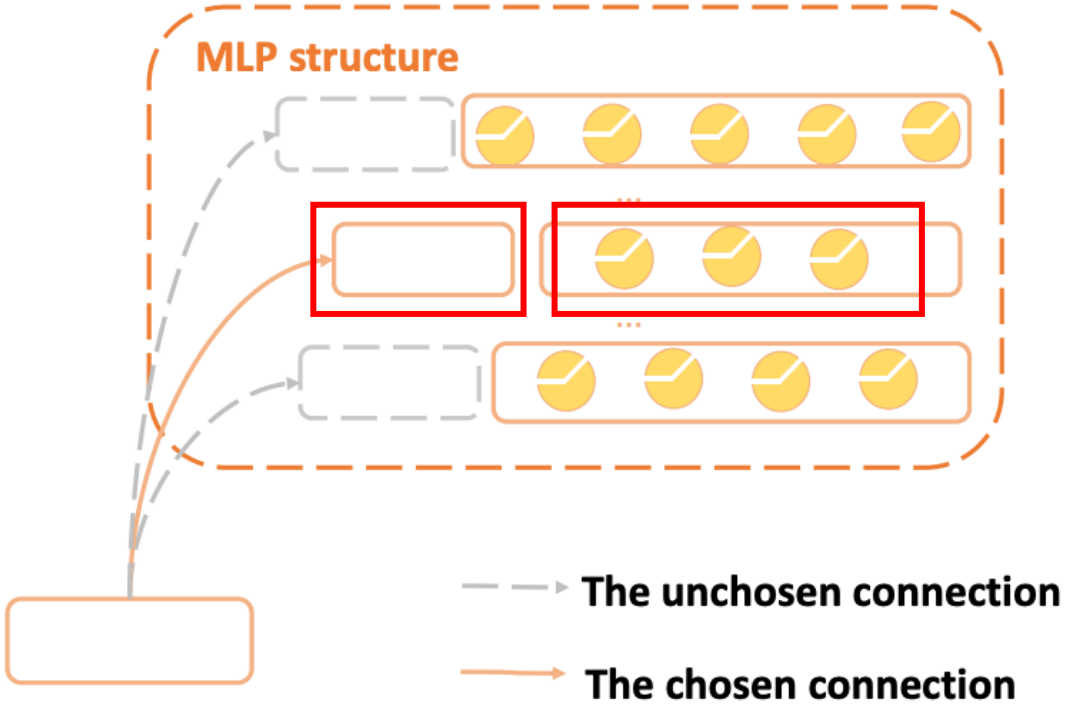
$$\{d_1, d_2, ..., d_n\}$$

- Feature interaction (S3)
  - First order features (N)
  - Second order interaction ($\binom{N}{2}$)

- Interaction function (S4)

$$f_{product} = \hat{e}_i \circ \hat{e}_j,$$
$$f_{concat} = Linear(concat[\hat{e}_i, \hat{e}_j])$$
$$f_{plus} = \hat{e}_i + \hat{e}_j,$$
$$f_{max} = max(\hat{e}_i, \hat{e}_j),$$

- MLP (Fixed L)
  - Input layer (S5) $\{1, 2, ..., L\}$
  - Layer dimensions (S6) $\{h_1, h_2, ..., h_n\}$



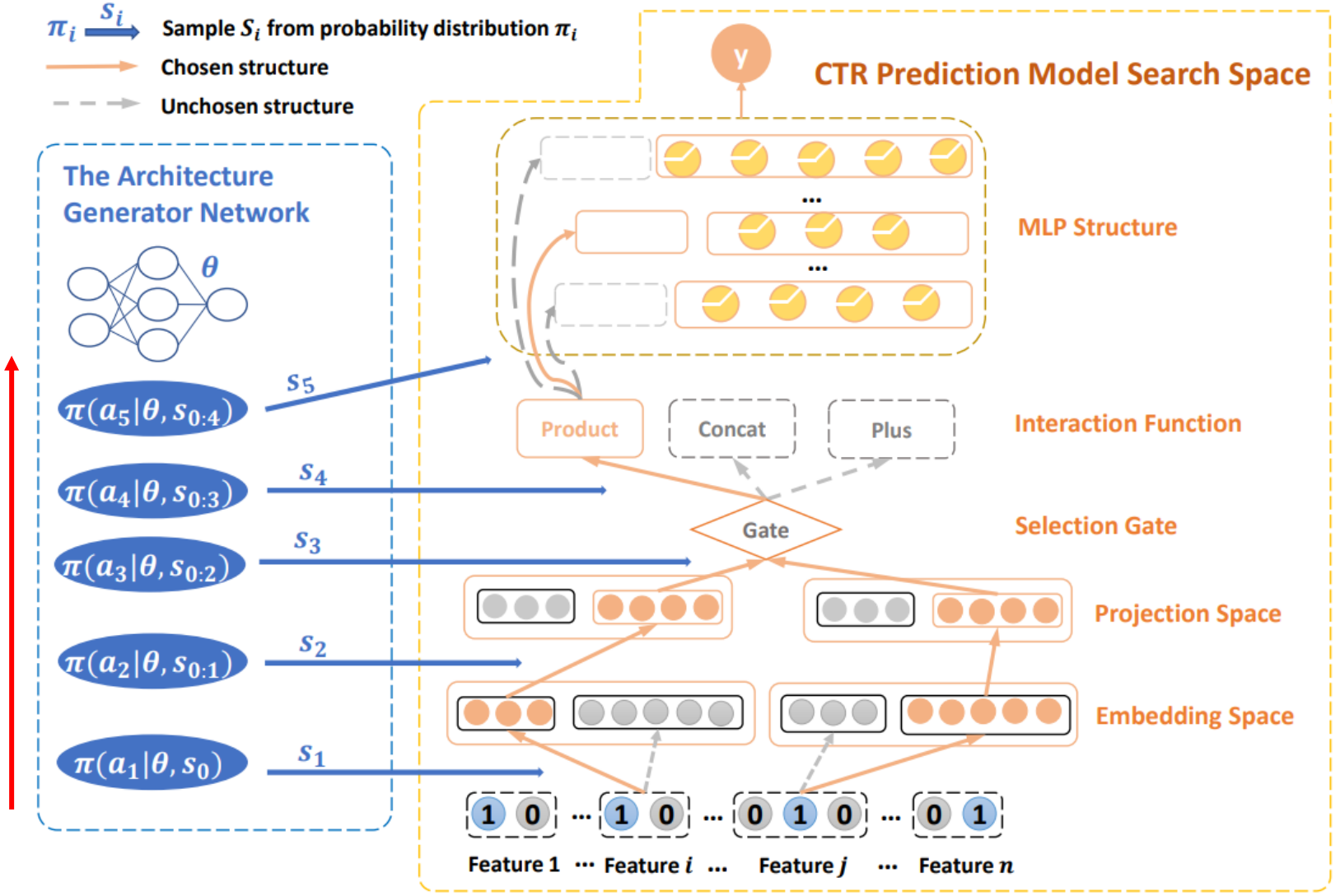
AutoIAS

- Sample process

$$P_i = \pi_i(\theta_i, a_{i-1}, ..., a_1),$$
$$a_i \sim P_i,$$

- Reward

$$R = \tau - \bar{\tau},$$



AutoIAS

- Step 1: Train teacher network (the largest one)

$$\mathcal{L}(y, y_t) = -y \log y_t - (1 - y) \log(1 - y_t)$$

- Step 2: Sample and update architectures as student network (KD step)

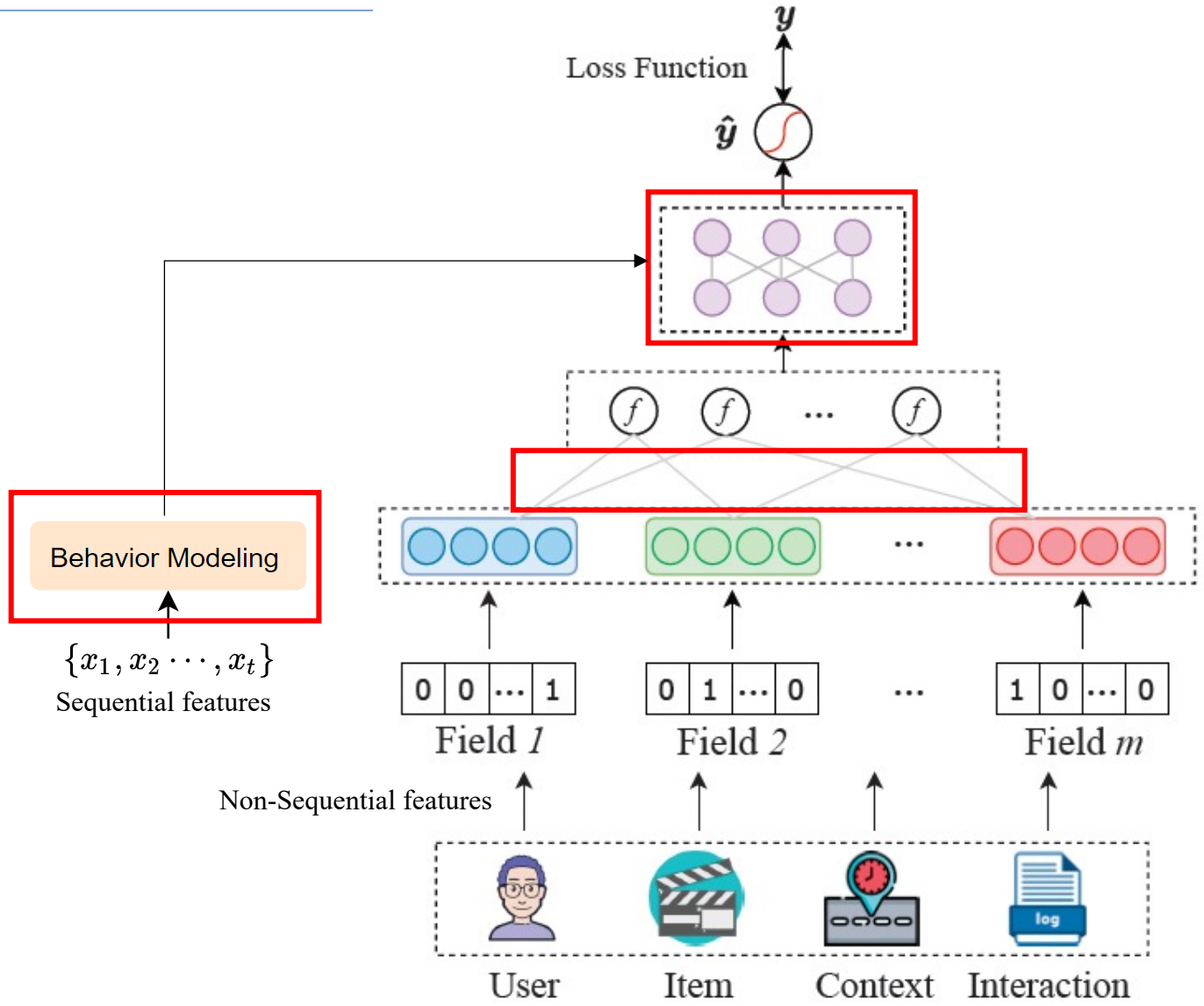
$$\mathcal{L}_{soft} = \mathcal{L}(\hat{y}_t, y_i) \quad \mathcal{L}_{hard} = \mathcal{L}(y, y_i) \quad \beta \mathcal{L}_{soft} + (1 - \beta) \mathcal{L}_{hard}$$

- Step 3: Sample architectures again and train the policy

$$\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta \log P(a_i; \theta) R_i$$

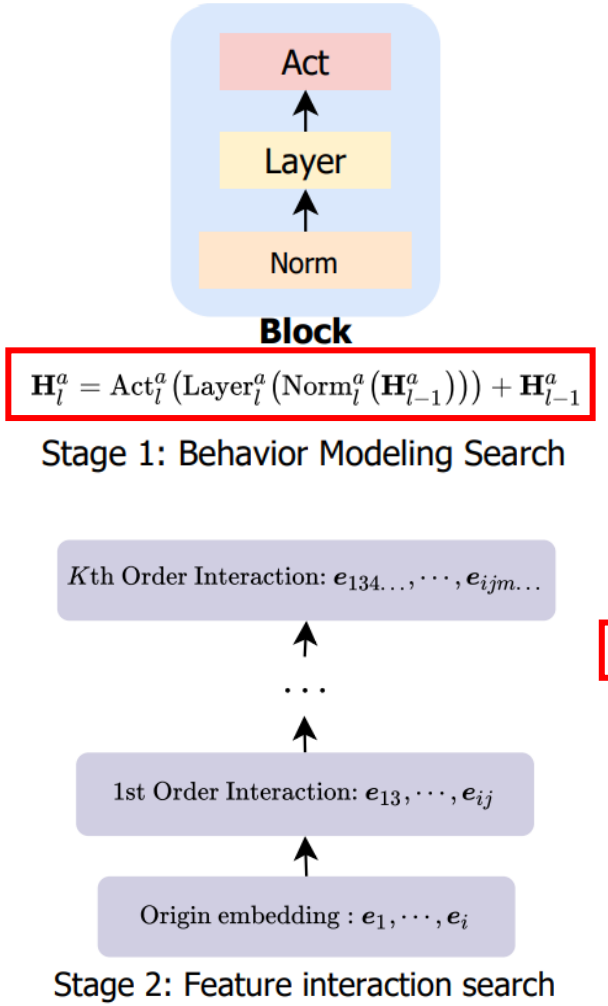
AMEIR

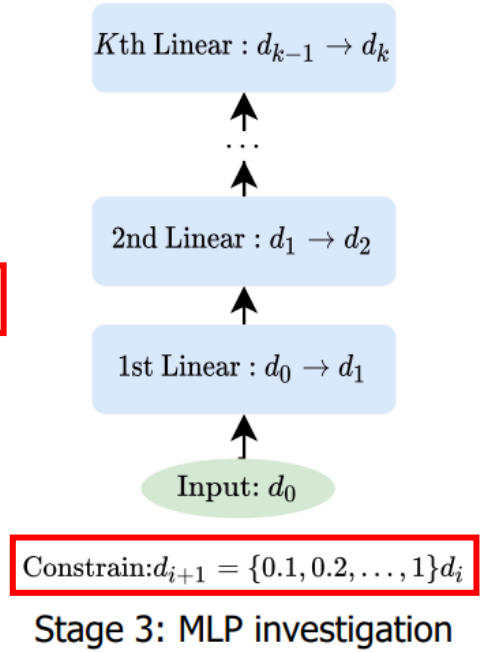
- Motivation:
  - 3 parts: sequential, non-sequential, MLP
  - Unified model for all scenarios
  - Restricted search space
- Target:
  - Searching for 3 parts
  - Adaptive model



AMEIR – Search Space

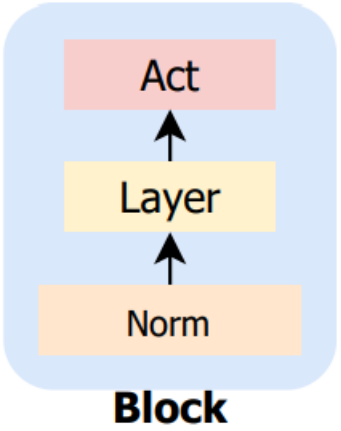
- Subspace 1 (Behavior modeling)
  - Searching for a fixed number of layers (L)
  - Normalization {Layer normalization, None}
  - Layer {Conv, Recur, Pooling, Attention}
  - Activation {ReLU, GeLU, Swish, Identity}
- Subspace 2 (Interaction exploration)
  - Interaction function: hadamard product (fixed)
  - Feature interaction candidates
- Subspace 3 (MLP investigation)
  - MLP dimension
  - Activation: {ReLU, Swish, Identity, Dice}





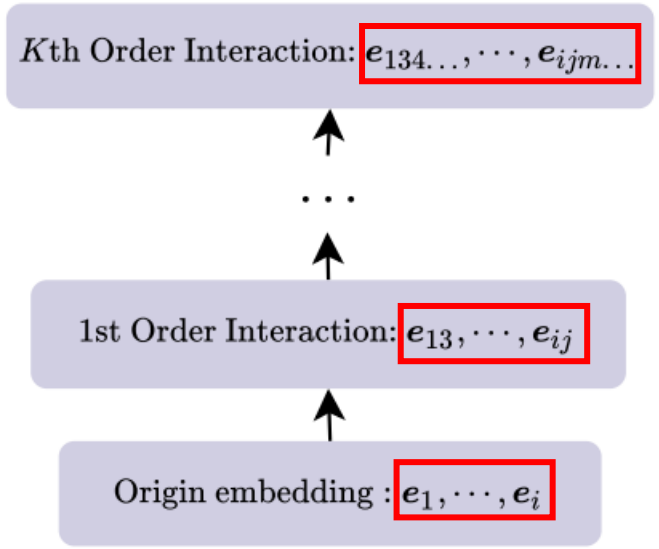
AMEIR

- Overall search strategy: One-shot random search
- Step 1: Using a predefined MLP, search for the optimal architecture.
- Step 2: Combined with SMBO, progressively expand the interaction sets, also use a predefined MLP.
- Step 3: Using a weight matrix of maximal dimension to realize one-shot search

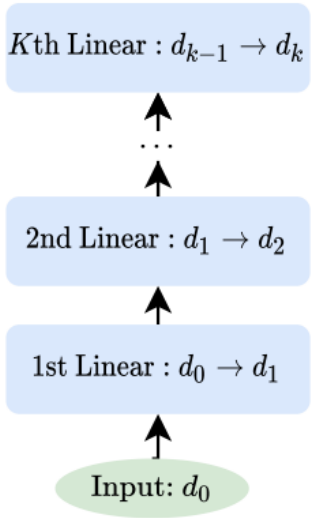


$$\mathbf{H}_l^a = \mathrm{Act}_l^a\left(\mathrm{Layer}_l^a\left(\mathrm{Norm}_l^a\left(\mathbf{H}_{l-1}^a\right)\right)\right) + \mathbf{H}_{l-1}^a$$

Stage 1: Behavior Modeling Search



Stage 2: Feature interaction search



Constrain: $d_{i+1} = \{0.1, 0.2, \ldots, 1\}d_i$

Stage 3: MLP investigation

Summarize DRS Comprehensive Search

Method	Search Space	Search Strategy
AIM	Embedding Dimension, Intearaction Function, Feature Interaction	Gradient
AutoIAS	Embedding Dimension, Projection Dimension, Interaction Function, Feature Interaction, MLP	Reinforcement Learning
AMEIR	Sequential model, Feature interaction, MLP	One-shot Random Search

- Existing comprehensive search methods mainly focus on feature embedding and feature interaction components. Some works also consider the MLP structure for final perdition while other parts as sequential feature modeling get few attention.
- The search space is very large for comprehensive search since it considers multiple components. As a result, efficient search strategies are usually adopted.

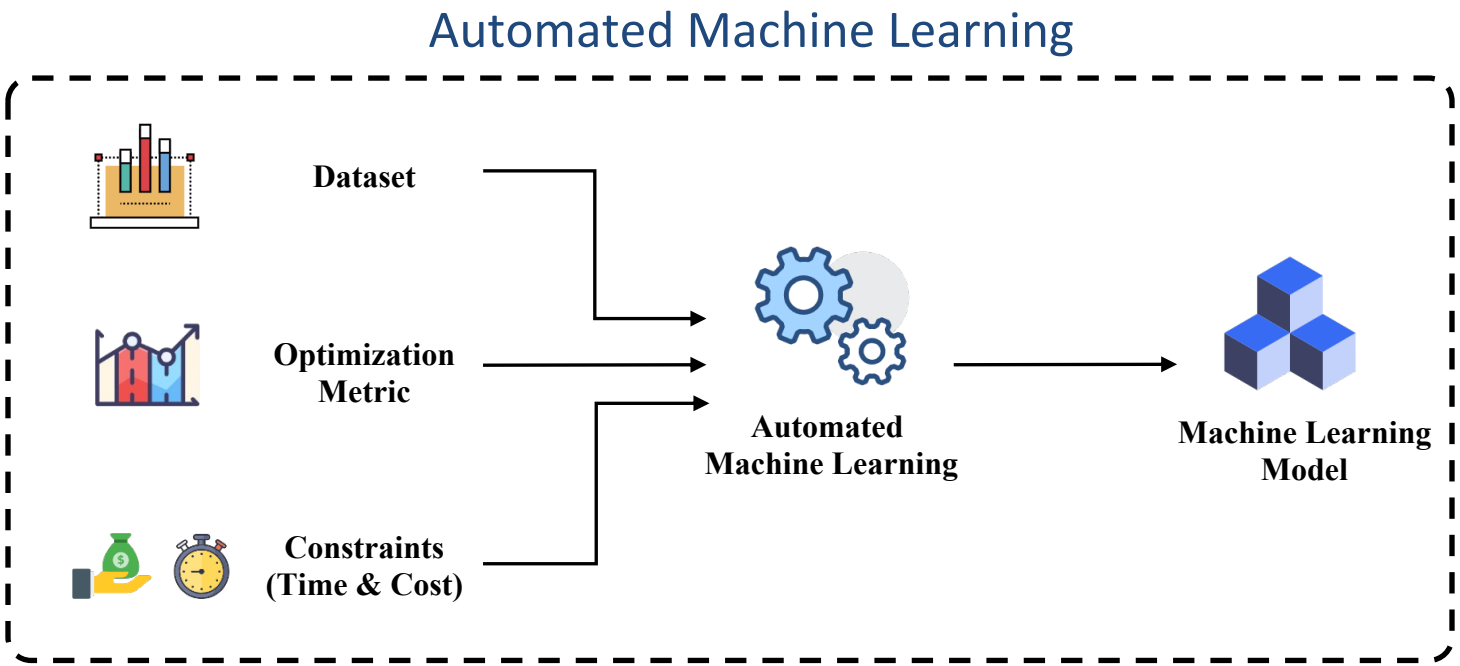
Table of Contents

- Introduction
- Preliminary of AutoML
- DRS Feature Selection
- DRS Embedding Components
- DRS Interaction Components
- DRS Model Training
- DRS Comprehensive Search
- Conclusion & Future Direction
- Q&A

Conclusion

Automated Machine Learning contribute to improving the performance of deep recommender systems in a data-driven manner.

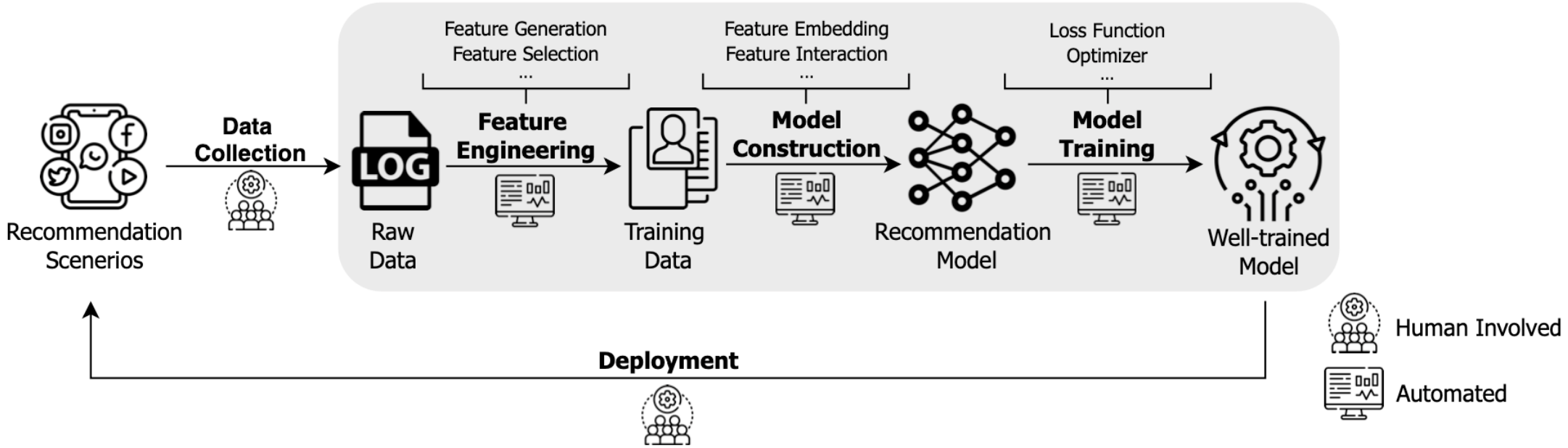
- Search embedding components to better model **feature representations**
- Design deep networks to better capture **feature interactions**
- Design **model training process** for more efficient and effective optimization
- Design **comprehensive system architectures** to better improve performance



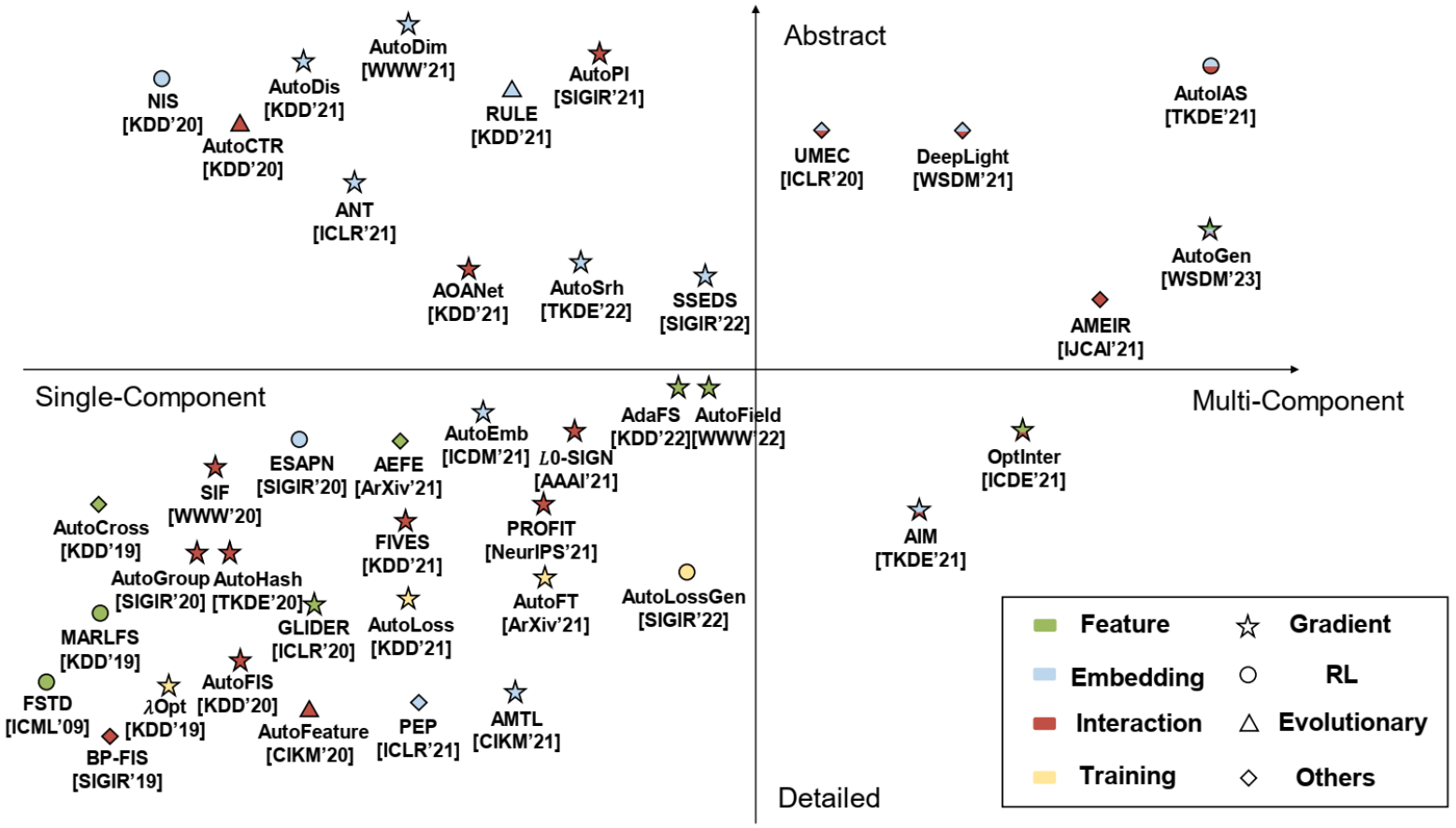
Conclusion

AutoML advantages:

- Different data → different architectures
- Less expert knowledge
- Saving time and efforts



AutoML for Deep Recommender Systems



The trend of AutoML for recommender system

- Existing AutoML-based work evolves from **single-component** search to **multi-component** joint search.
- The search space of these AutoML-based work develops from **detailed** to **abstract** for shrinking search space and improving search efficiency.
- The search algorithm of existing work is mainly based on gradient-based methods, thus providing **efficient model searching and training mode**.

Summarize DRS Feature Selection

Model	Candidates	Granularity	Combination	Gating/Scoring	Search Strategy
FSTD	Raw features	Field-level	Filter/Wrapper	AUC	RL
MARLFS	Raw features	Field-level	Embedded	None	RL
AutoField	Raw features	Field-level	Embedded	Continuous	Gradient
AdaFS	Raw features	Field-level	Embedded	Continuous	Gradient
LPFS	Raw features	Field-level	Embedded	Zero/Non-zero	Gradient
OptFS	Raw features	Feature-level	Embedded	Approx. zero	Gradient
AutoCross	Generated features	Field-level	Wrapper	AUC	Beam Search
GLIDER	Generated features	Field-level	Filter	NID	Gradient
AEFE	Generated features	Field-level	All	Continuous	Greedy Search

Selection of raw features

Model	Granularity	Gating/Scoring	Search Strategy
FSTD	Field-level	Temporal Difference	RL (TD + UCB)
MARLFS	Field-level	None	RL (DQN)
AutoField	Field-level	Continuous	Gradient
AdaFS	Field-level	Continuous	Gradient
LPFS	Field-level	Zero/Non-zero	Gradient (L0)
OptFS	Feature-level	Approx. zero	Gradient (L0)

- Reinforcement learning methods consider the problem of feature selection as a Markov decision process. They are less prone to overfitting since they usually overall performance of the model when designing reward functions;

- Gradient-based approaches are more practical to real-world recommender systems owing to their efficiency and simplicity. In addition, they are flexible to be applied to various recommendation models and datasets.

Selection of generated features

Model	Combination	Granularity	Gating/Scoring	Search Strategy
AutoCross	Wrapper	Field-level	None	Beam Search
GLIDER	Filter	Field-level	NID	Gradient
AEFE	All	Field-level	Continuous	Greedy Search

- Selectively learning the generated features can bring great precision improvement to prediction. They are highly interpretable, which is helpful for digging deep into the underlying relationship of the data;
- Compared with selection from raw features, generated feature selection usually has a much larger search space. Researchers usually adopt greedy search methods, which suffer from the heavy storage pressure and time-consuming process. It is highly desirable for efficient AutoML techniques to facilitate the selection from generated features.

Future Direction for Feature Selection

Model	Candidates	Granularity	Combination	Gating/Scoring	Search Strategy
FSTD	Raw features	Field-level	Filter/Wrapper	AUC	RL
MARLFS	Raw features	Field-level	Embedded	None	RL
AutoField	Raw features	Field-level	Embedded	Continuous	Gradient
AdaFS	Raw features	Field-level	Embedded	Continuous	Gradient
LPFS	Raw features	Field-level	Embedded	Zero/Non-zero	Gradient
OptFS	Raw features	Feature-level	Embedded	Approx. zero	Gradient
AutoCross	Generated features	Field-level	Wrapper	AUC	Beam Search
GLIDER	Generated features	Field-level	Filter	NID	Gradient
AEFE	Generated features	Field-level	All	Continuous	Greedy Search

1) Feature Selection

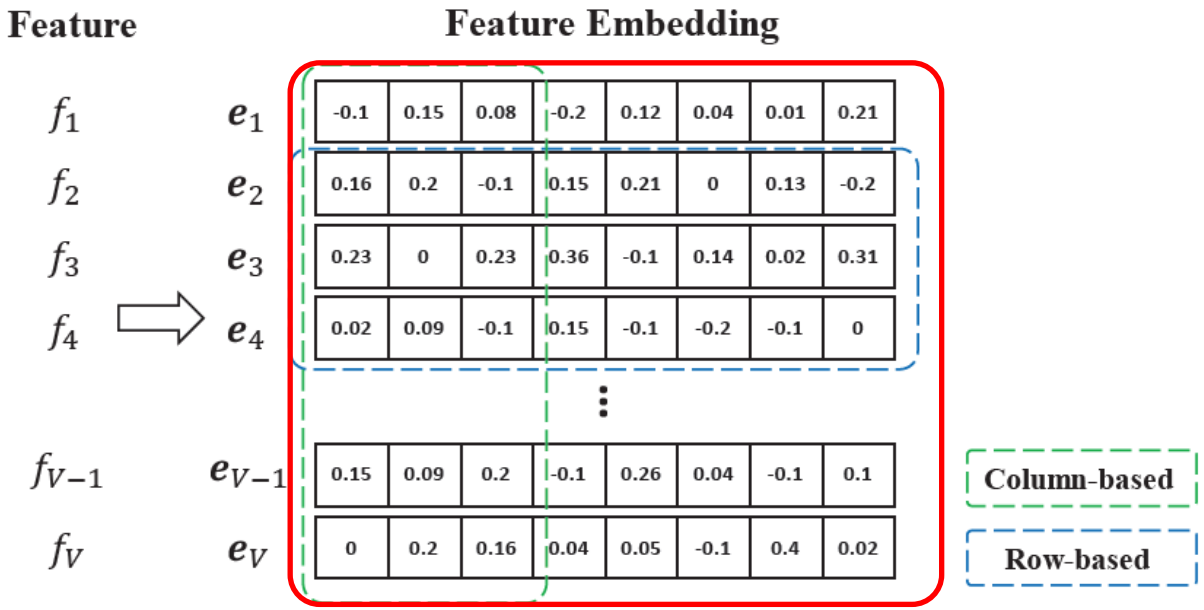
- **Combinatorial features** are of great importance for recommender systems.
- How to **generate and select combinatorial features effectively and save memory usage** is an urgent problem for both industry and academics

Summarize DRS Feature Embedding

Method	Column-Wise	Row-Wise	Search Space	Multi-Embedding	Search Algorithm	Memory Reduction
AMTL	×	×	d^V	×	Gradient	×
PEP	×	×	2^{Vd}	×	Regularization	×
AutoEmb	√	×	d^V	√	Gradient	×
ESAPN	√	×	a^V	√	RL	×
SSEDS	×	√	2^{md}	×	Gradient	√
AutoSrh	×	√	2^{bd}	×	Gradient	√
AutoDim	√	√	a^m	×	Gradient	√
NIS	√	√	ab *or* b^a	×	RL	√
RULE	√	√	2^{ab}	×	Evolutionary	√
ANT	-	-	2^{kV}	×	Gradient	√
AutoDis	-	-	2^{kV}	×	Gradient	√

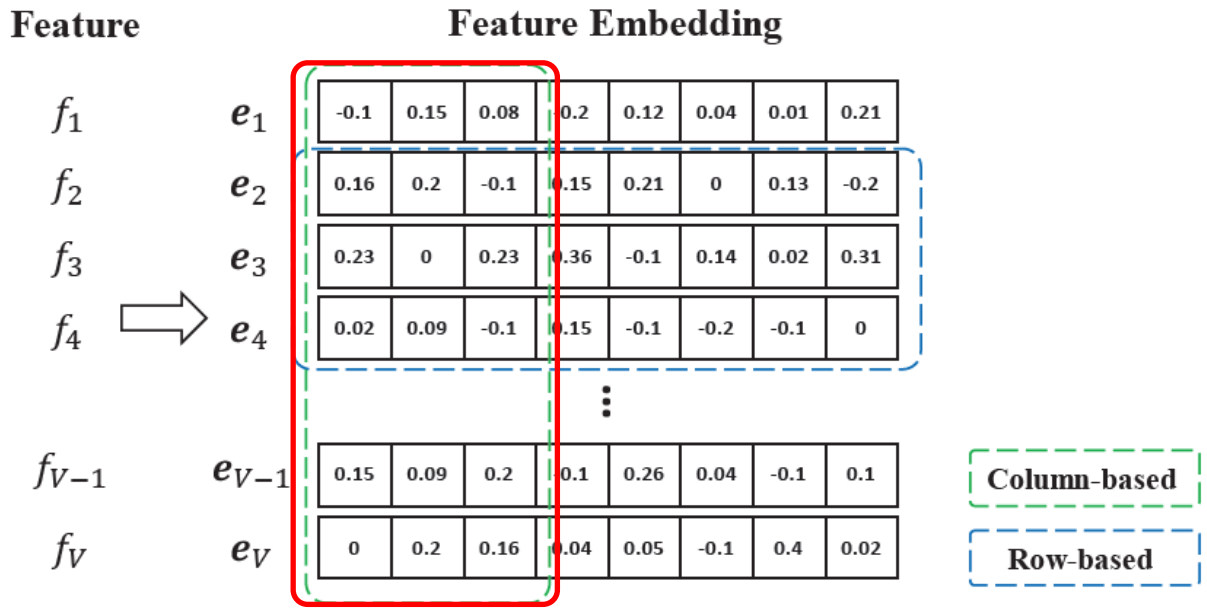
* d is the embedding size, V is the vocabulary size, m is the number of feature fields, a is the number of sub-dimensions, b is the number of feature groups, k is the number of meta-embeddings. (a < d, b<<V)

Full Embedding Search——Summary



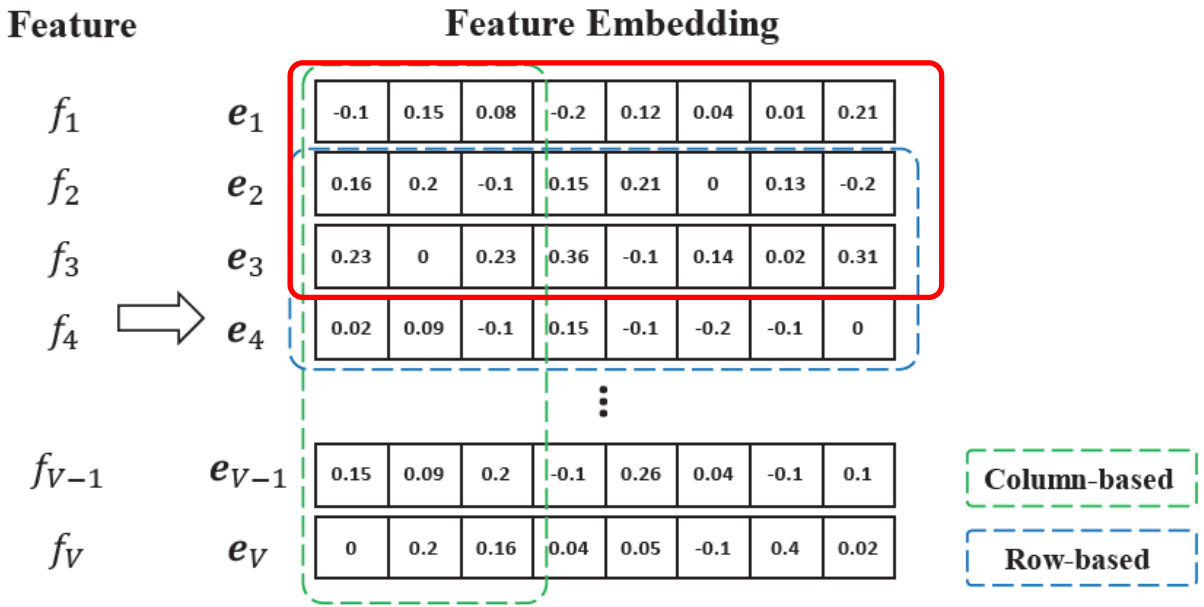
- Full embedding search methods aim to search the optimal embedding dimension for each feature value, facing **huge search space** and impeding the search efficiency;
- To facilitate the search procedure, several approaches are proposed to **shrink the search space**, which can be categorized into three kinds: column-based, row-based, and column & row-based.

Column-based Embedding Search——Summary



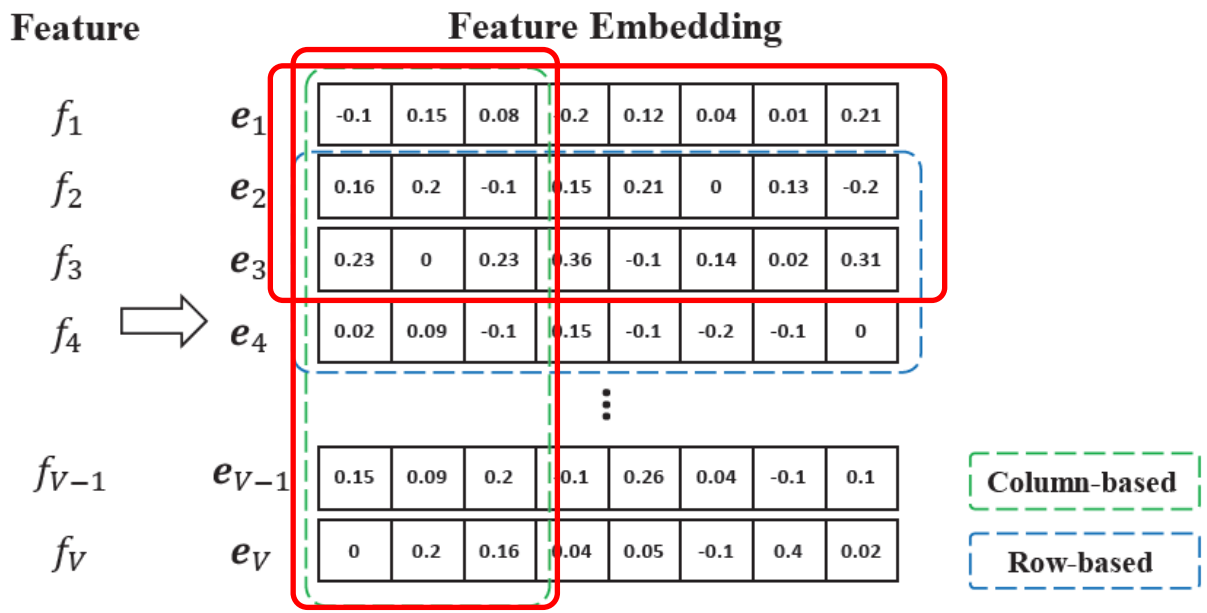
- Dividing the embedding dimension into column-wise sub-dimensions (e.g., AutoEmb, ESAPN) is conducive to **reducing the search space**;
- Using multiply embedding tables to generate several embedding vectors (e.g., AutoEmb, ESAPN) may incur obvious memory overhead, which can be avoid by shared-embeddings;
- Searching dimensions for each feature value will cause variable-length embedding vectors, which are **hard to store** in the fix-width embedding table and reduce memory.

Row-based Embedding Search——Summary



- Row-based embedding search methods explore optimal embedding dimension for a group of feature values, **shrinking the search space**;
- In comparison with the column-based embedding search methods, row-based search methods conduces to **truly saving memory** because feature values within a group are assigned with a same embedding dimension, which can be stored in a fix-width embedding table.

Column & Row-based Embedding Search——Summary



- Although it is theoretically optimal to search the suitable dimension for each feature value, it poses great challenges to efficient search algorithm. Instead, shrinking the search space in an appropriate manner may result in better performance;
- Reducing the search space from both row-wise and column-wise perspectives attributes to reducing the search space and achieving better results;
- The evolution of search space **from detailed to abstract** can lead to higher efficiency.

Future Direction for Feature Embedding

Method	Column-Wise	Row-Wise	Search Space	Multi-Embedding	Search Algorithm	Memory Reduction
AMTL	×	×	d^V	×	Gradient	×
PEP	×	×	2^{Vd}	×	Regularization	×
AutoEmb	√	×	d^V	√	Gradient	×
ESAPN	√	×	a^V	√	RL	×
SSEDS	×	√	2^{md}	×	Gradient	√
AutoSrh	×	√	2^{bd}	×	Gradient	√
AutoDim	√	√	a^m	×	Gradient	√
NIS	√	√	ab or b^a	×	RL	√
RULE	√	√	2^{ab}	×	Evolutionary	√
ANT	-	-	2^{kV}	×	Gradient	√
AutoDis	-	-	2^{kV}	×	Gradient	√

2) Feature Embedding Search

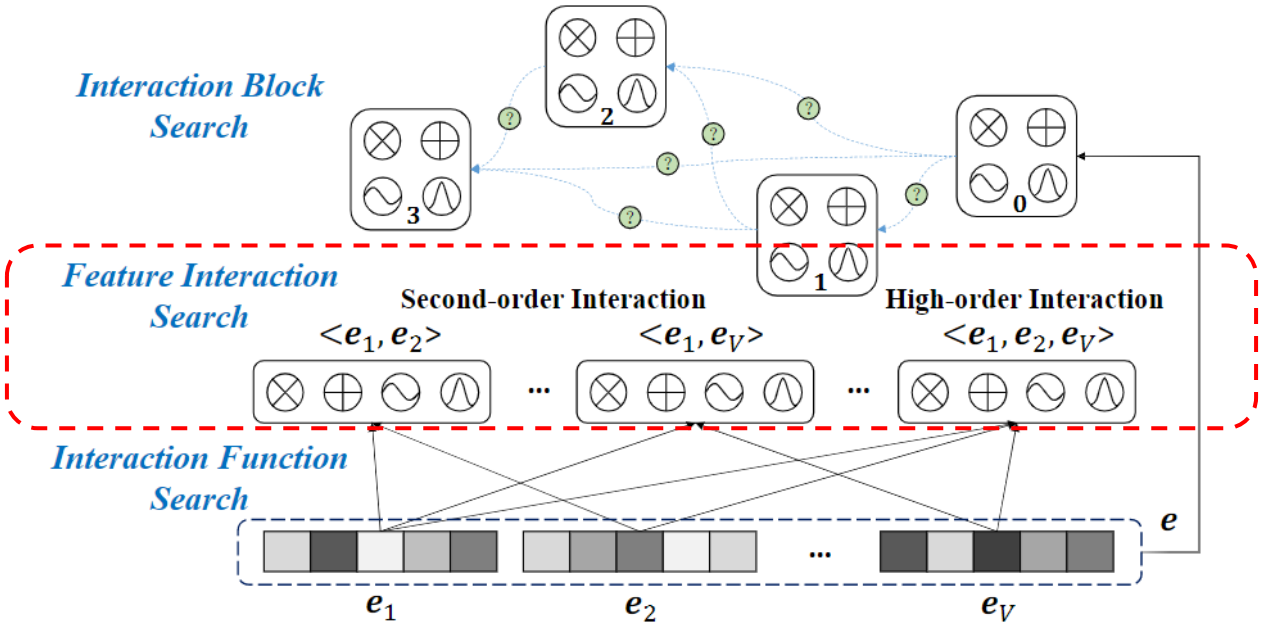
- Feature embeddings account for the majority of the parameters for the recommendation model
- Combining the **feature representation learning** with **model compression or quantization** automatically may be a promising research direction

Summarize DRS Feature Interaction

Method	Search Category	Search Space	Order	Search Algorithm
AutoFIS	Feature Interaction	$2^{C_m^p}$	second-order	Gradient
AutoGroup	Feature Interaction	2^{gm}	high-order	Gradient
PROFIT	Feature Interaction	mR	high-order	Gradient
FIVES	Feature Interaction	2^{m^2}	high-order	Gradient
L_0-SIGN	Feature Interaction	2^{m^2}	second-order	Gradient
BP-FIS	Feature Interaction	$u2^{C_m^p}$	second-order	Bayesian
SIF	Interaction Function	C_c^n	second-order	Gradient
AutoFeature	Interaction Function	$c^n n C_m^p$	second-order	Evolutionary
AOANet	Interaction Function	$2^{\lvert(i,j)\ pairs\rvert}$	high-order	Gradient
AutoCTR	Interaction Block	$c^{C_n^2}$	high-order	Evolutionary
AutoPI	Interaction Block	$c^{C_n^2}$	high-order	Gradient

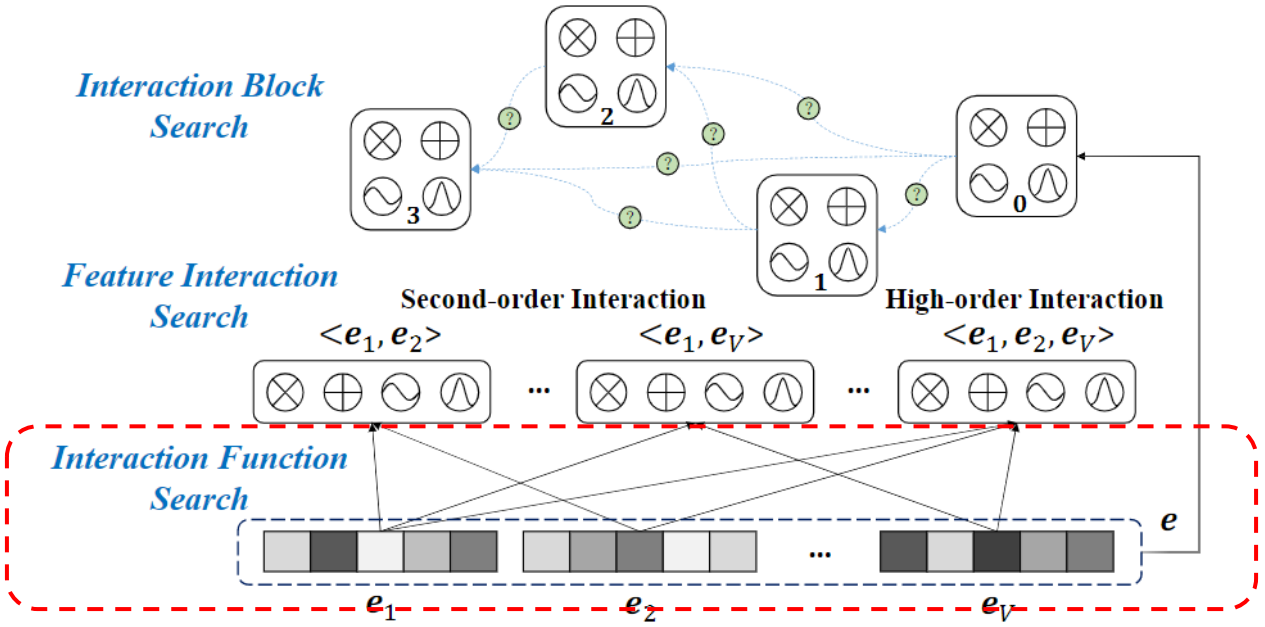
* m is the number of feature fields, p is the order, g is the number of pre-defined groups, n is the number of pre-defined bolcks, c is the number of candidate interaction functions.

Feature Interaction Search——Summary



- Feature interaction search based methods focus on searching beneficial low-/high order interactive signals for factorization models;
- For high-order interaction search, different approaches are proposed to reduce the search space, such as feature grouping, hashing, tensor decomposition, and graph aggregation;
- Gradient-based search algorithm is dominant in this task due to the high efficiency.

Interaction Function Search——Summary



- Although searching appropriate interaction functions for different feature interactions helps to improve accuracy, the introduced cost is higher, which hinders the application in high-order scenarios;
- Generalized interaction function search (e.g., AOANet) is more efficient than searching over human-designed search space (e.g., AutoFeature), providing a promising paradigm for high-order interaction function search.

Summarize DRS Interaction

Method	Search Category	Search Space	Order	Search Algorithm
AutoFIS	Feature Interaction	$2^{C_m^p}$	second-order	Gradient
AutoGroup	Feature Interaction	2^{gm}	high-order	Gradient
PROFIT	Feature Interaction	mR	high-order	Gradient
FIVES	Feature Interaction	2^{m^2}	high-order	Gradient
L_0-SIGN	Feature Interaction	2^{m^2}	second-order	Gradient
BP-FIS	Feature Interaction	$u2^{C_m^p}$	second-order	Bayesian
SIF	Interaction Function	C_c^n	second-order	Gradient
AutoFeature	Interaction Function	$c^n nC_m^p$	second-order	Evolutionary
AOANet	Interaction Function	$2^{\lvert(i,j)\ pairs\rvert}$	high-order	Gradient
AutoCTR	Interaction Block	$c^{C_n^2}$	high-order	Evolutionary
AutoPI	Interaction Block	$c^{C_n^2}$	high-order	Gradient

* m is the number of feature fields, p is the order, g is the number of pre-defined groups, n is the number of pre-defined bolcks, c is the number of candidate interaction functions.

Future Direction for Feature Interaction

Method	Search Category	Search Space	Order	Search Algorithm
AutoFIS	Feature Interaction	$2^{C_m^p}$	second-order	Gradient
AutoGroup	Feature Interaction	2^{gm}	high-order	Gradient
PROFIT	Feature Interaction	mR	high-order	Gradient
FIVES	Feature Interaction	2^{m^2}	high-order	Gradient
L_0-SIGN	Feature Interaction	2^{m^2}	second-order	Gradient
BP-FIS	Feature Interaction	$u2^{C_m^p}$	second-order	Bayesian
SIF	Interaction Function	C_c^n	second-order	Gradient
AutoFeature	Interaction Function	$c^n nC_m^p$	second-order	Evolutionary
AOANet	Interaction Function	$2^{\lvert(i,j)\ pairs\rvert}$	high-order	Gradient
AutoCTR	Interaction Block	$c^{C_n^2}$	high-order	Evolutionary
AutoPI	Interaction Block	$c^{C_n^2}$	high-order	Gradient

3) Feature Interaction Search

- Existing interaction functions, e.g., inner product and MLP, are widely-used for recommendation
- Designing and introducing **more informative interaction operators** to generate **more diverse interaction functions** may improve the model prediction performance

Summarize DRS Model Training

Method	Search Space	Search Strategy
AutoLoss	Optimization: Loss Function	Gradient
AutoLossGen	Optimization: Loss Function	RL
AutoFT	Parameter Tuning: Fine-Tune or Not	Gradient

- Loss-based optimization methods facilitate recommendation model training via searching optimal loss function automatically, bringing significant results;
- From loss function to transfer learning, researchers gradually realize more flexible and efficient methodologies to facilitate model training.

Summarize DRS Model Training

Method	Search Space	Search Strategy
AutoLoss	Optimization: Loss Function	Gradient
AutoLossGen	Optimization: Loss Function	RL
AutoFT	Parameter Tuning: Fine-Tune or Not	Gradient

- Loss-based optimization methods facilitate recommendation model training via searching optimal loss function automatically, bringing significant results;
- From loss function to transfer learning, researchers gradually realize more flexible and efficient methodologies to facilitate model training.

Future Direction for Model Training

Method	Search Space	Search Strategy
AutoLoss	Optimization: Loss Function	Gradient
AutoLossGen	Optimization: Loss Function	RL
AutoFT	Parameter Tuning: Fine-Tune or Not	Gradient

4) Model Training

- Existing works mainly focus on the **training loss**, including loss function selection or generation and regularization adjustment.
- More complex directions for model training should be explored, e.g**., optimizer settings, gradient direction guidance**.

Summarize DRS Comprehensive Search

Method	Search Space	Search Strategy
AIM	Embedding Dimension, Intearaction Function, Feature Interaction	Gradient
AutoIAS	Embedding Dimension, Projection Dimension, Interaction Function, Feature Interaction, MLP	Reinforcement Learning
AMEIR	Sequential model, Feature interaction, MLP	One-shot Random Search

- Existing comprehensive search methods mainly focus on feature embedding and feature interaction components. Some works also consider the MLP structure for final perdition while other parts as sequential feature modeling get few attention.
- The search space is very large for comprehensive search since it considers multiple components. As a result, efficient search strategies are usually adopted.

Future Direction for Comprehensive Search

Method	Search Space	Search Strategy
AIM	Embedding Dimension, Intearaction Function, Feature Interaction	Gradient
AutoIAS	Embedding Dimension, Projection Dimension, Interaction Function, Feature Interaction, MLP	Reinforcement Learning
AMEIR	Sequential model, Feature interaction, MLP	One-shot Random Search

5) Comprehensive Search

- Existing solutions **search each component separately** with **heterogeneous search space**, resulting in low search efficiency and sub-optimal performance.
- It is potential to **convert the heterogeneous search space into an isomorphic unified search space** and perform efficient search algorithm.

Future Directions

6) Multi-task Learning

- **Exploiting different revenue targets** (e.g., click-through rate and conversion rate), is one of the most important techniques for industry recommendations. It is worthy of **designing an automatic algorithm for adaptive multi-task learning**.

7) User Behavior Modeling

- User history behaviors contain different dimensions of interests. **Automatically retrieving beneficial history behaviors** for modeling user preference is an important further direction.

Q&A

A Comprehensive Survey on Automated Machine Learning for Recommendations.

https://arxiv.org/pdf/2204.01390