pykt-toolkit

Release 0.0.37

pykt-team

Mar 09, 2024

HOME

pyKT is a python library build upon PyTorch to train deep learning based knowledge tracing (KT) models. The library consists of a standardized set of integrated data preprocessing procedures on multi popular datasets across different domains, 5 detailed prediction scenarios, frequently compared DLKT approaches for transparent and extensive experiments.

Let's Get Started! English Introduction.

More details about the academic information can be read in our paper at https://arxiv.org/abs/2206.11460?context=cs.CY .

CHAPTER

ONE

QUICK START

1.1 Installation

You can specify to install it through `pip`.

```
pip install -U pykt-toolkit
```

We advise to create a new Conda environment with the following command:

```
conda create --name=pykt python=3.7.5
source activate pykt
pip install -U pykt-toolkit
```

1.2 Train Your First Model

1.2.1 Prepare a Dataset

1Obtain a Dataset

Let's start by downloading the dataset from *here*. Please make sure you have creat the `data/{dataset_name}` folder

2Data Preprocessing

`python data_preprocess.py [parameter]`

```
Args:
    --dataset_name: dataset name, default="assist2015"
    --min_seq_len: minimum sequence length, default=3
    --maxlen: maximum sequence length, default=200
    --kfold: divided folds, default=5
```

Example:

```
cd examples
python data_preprocess.py --dataset_name=ednet
```

1.2.2 Training a Model

After the data preprocessing, you can use the `python wandb_modelname_train.py [parameter]` to train a model:

```
CUDA_VISIBLE_DEVICES=2 nohup python wandb_sakt_train.py --dataset_name=assist2015 --use_
↪wandb=0 --add_uuid=0 --num_attn_heads=2 > sakt_train.txt &
```

1.3 Evaluating Your Model

Now, let's use `wandb_predict.py` to evaluate the model performance on the testing set.

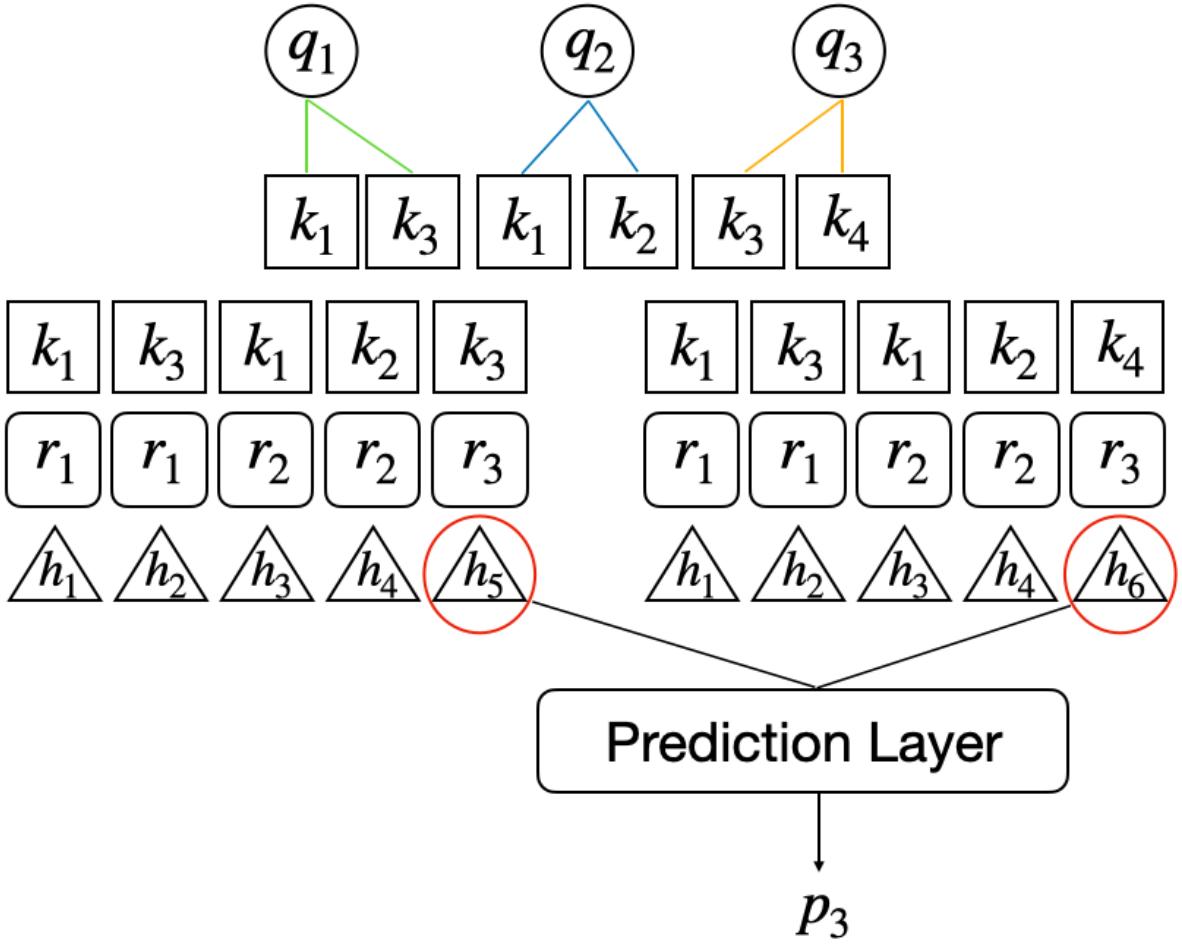
`python wandb_predict.py`

```
Args:
    --bz: batch_size, default is 256
    --save_dir: the dictory of the trained model, default is "saved_model"
    --fusion_type: the fusion mode,default is "late_fusion"
    --use_wandb: use wandb or not, default is 1
```

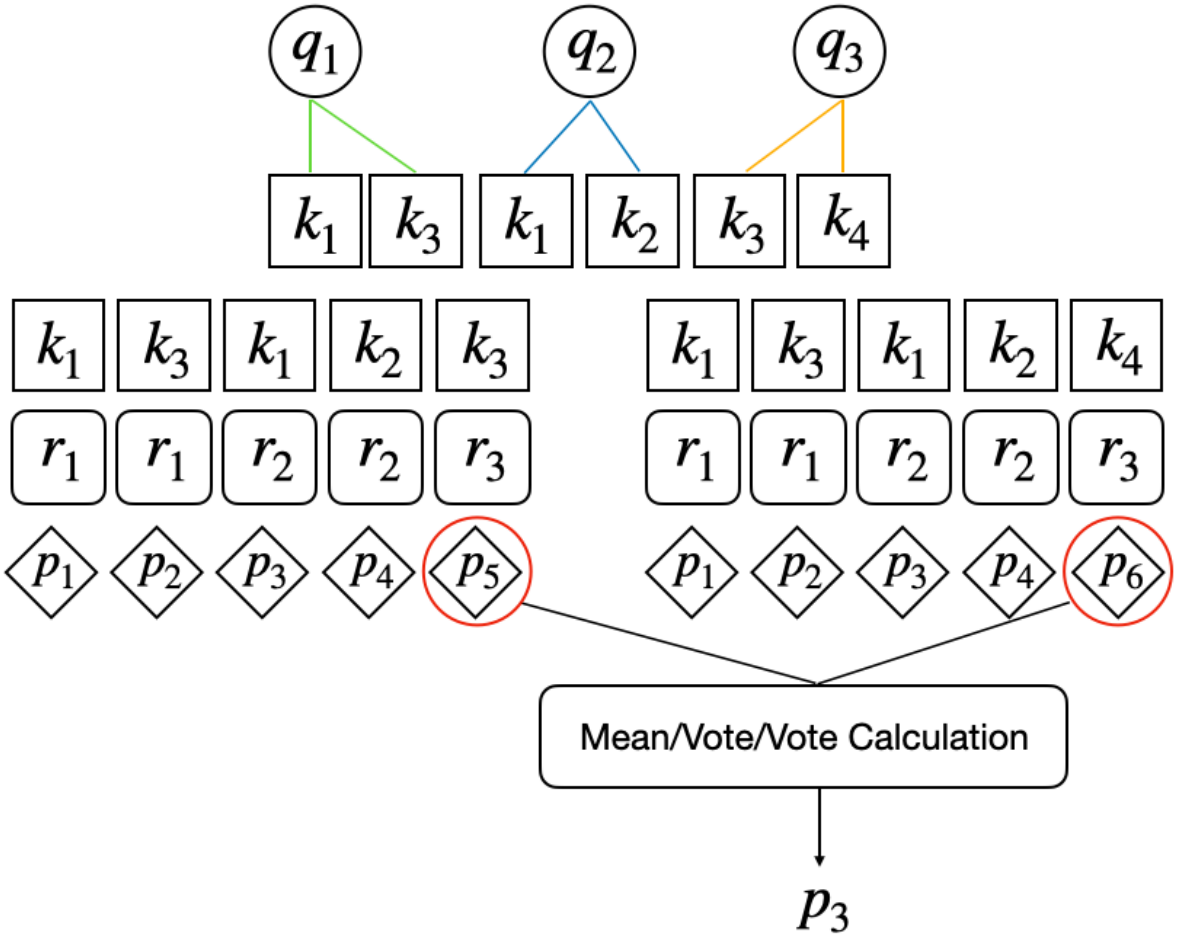
1.3.1 Evaluation Protocol

A question may be related to multiple knowledge concepts (KCs). To make the evaluation of pyKT is consistent with the real-world prediction scenarios, we train DLKT models on KCs but evaluate them on questions level as follows:

- **Early fusion:** Calculate the average of the hidden states on KC levels, and then input the average results into the prediction layer, hence get the prediction results on question level. For example, to obtain the prediction p_3 of q_3, we average the hidden states h_5,h_6 into the prediction layer.



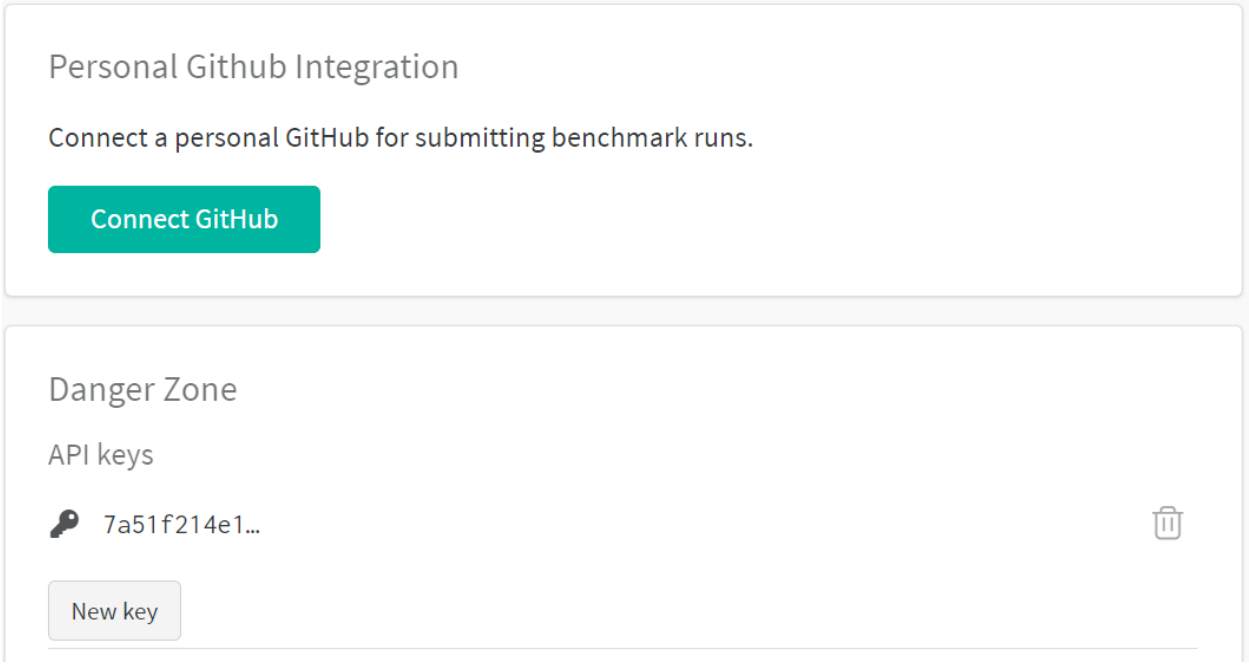
- **Late fusion:** Employ three fusion types to obtain the question-level prediction based on the KC-level prediction results:(1) *Mean*: compute the average of the KC-level prediction results as the final prediction. (2) *Vote*: select half of the values of KC predictions as the final prediction. (3) *All*: only if all KC predictions are correct, the final prediction is correct, otherwise it is wrong.



1.4 Hyperparameter Tuning

1.4.1 Create a Wandb Account

We use Weights & Biases (Wandb) for hyperparameter tuning, it is a machine learning platform for developers to build better models faster with experiment tracking. Firstly, let's register an account in Wandb webpage to get the API key from here:



Next, add your `uid` and `api_key` into `configs/wandb.json`.

1.4.2 Sweep Configuration

`[wandb_key] python generate_wandb.py [parameter]`

```
Args:
       --src_dir: The parameter configuration file path of the model
       --project_name: Project name on wandb, default: kt_toolkits
       --dataset_names: Dataset names, you can fill in multiple, separated by commas ",",
↪ default: "assist2015"
       --model_names: Model names, you can fill in multiple, separated by commas ",",␣
↪default: dkt
       --emb_type: Default:qid
       --folds: Default: "0,1,2,3,4"
       --batch_size: Default: 128
       --save_dir_suffix: Add extra characters to the model storage path name, default: "
↪"
       --all_dir: Generate the configuration file of the model for this dataset,␣
↪default: "all_wandbs"
       --launch_file: Generated sweep startup script, default: "all_start.sh"
       --generate_all: The input is "True" or "False", indicating whether to generate␣
↪the wandb startup files of all datasets and models in the all_dir directory (True␣
↪means: generate the startup files of all data models in the all_dir directory, False␣
↪means: only the current execution is generated data model startup file), default:
↪"False"
```

Example:

```
WANDB_API_KEY=xxx python generate_wandb.py --dataset_names "assist2009,assist2015"  --
↪project_name hawkes --model_names "dkt,dkt+"
```

1.4.3 Start Sweep

Step1: sh [launch_file] [parameter]

```
sh [launch_file] > [Directed log] 2>&1

    - [launch_file]: required, the user submits the script of sweep to wandbs, and␣
↪directs the execution output to [directed log])
    - [Directed log]: Required, execute the sweep in the log
```

Example:

```
sh all_start.sh > log.all 2>&1
(You need to define the log file. )
```

Step 2: sh run_all.sh [parameter]

```
[wandb_key] sh run_all.sh [Directed log] [start_sweep] [end_sweep] [dataset_name] [model_
↪name] [gpu_ids] [project_name]

    - [Directed log]: Required, execute the sweep in the log
    - [start_sweep]: Required, the start id to start a sweep
    - [end_sweep]: Required, start sweep end id
    - [dataset_name]: Required, dataset name
    - [model_name]: Required, model name
    - [gpu_ids]: Required, GPU ID
    - [project_name]: optional, default: kt_toolkits
```

Example:

```
WANDB_API_KEY=xxx sh run_all.sh log.all 0 5 assist2009 dkt 0,1,2,3,4 nips2022-assist2009
```

1.4.4 Start Agents

```
sh start_sweep_0_5.sh
("0", "5" denote the start sweep and end sweep respectively.)
```

1.4.5 Tuning Protocol

We use the Bayes search method to find the best hyperparameter, it is expensive to run all the hyperparameter combinations. Hence, you can run the `pykt-toolkit/examples/check_wandb_status.ipynb` file to check whether to stop the searching. We default to stop the searching if the number of the tuned hyperparameter combinations in each data fold is larger than 200 and there is no AUC improvement on the testing data in the last 100 rounds (output "end!").

1.4.6 Start Evaluation

- Extract best model

```
def extract_best_models(self, df, dataset_name, model_name, emb_type="qid", eval_
↪test=True, fpath="./seedwandb/predict.yaml", CONFIG_FILE="../configs/best_model.json",␣
↪wandb_key="", pred_dir="pred_wandbs", launch_file="start_predict.sh", generate_
↪all=False):
    """extracting the best models which performance best performance on the validation␣
↪data for testing

    Args:
        df: dataframe of best results in each fold
        dataset_name: dataset_name
        model_name: model_name
        emb_type: embedding_type, default:qid
        eval_test: evaluating on testing set, default:True
        fpath: the yaml template for prediction in wandb, default: "./seedwandb/
↪predict.yaml"
        config_file: the config template of generating prediction file, default: "../
↪configs/best_model.json"
        wandb_key: the key of wandb account
        pred_wandbs: the directory of prediction yaml files, default: "pred_wandbs"
        launch_file: the launch file of starting the wandb prediction, default: "start_
↪predict.sh"
        generate_all: starting all the files on the pred_wandbs directory or not,␣
↪default:False

    Returns:
        the launch file (e.g., "start_predict.sh") for wandb prediction of the best␣
↪models in each fold
    """
    if not os.path.exists(pred_dir):
        os.makedirs(pred_dir)
    model_path_fold_first = []
    dconfig = dict()
    for i, row in df.iterrows():
        fold, model_path = row["fold"], row["model_save_path"]
        model_path = model_path.rstrip("qid_model.ckpt")
        print(f">>> The best model of {dataset_name}_{model_name}_{fold}:{model_path}")
        model_path_fold_first.append(model_path)
    ftarget = os.path.join(pred_dir, "{}_{}_{}_fold_first_predict.yaml".format(dataset_
↪name, model_name, emb_type))
    if eval_test:
        self.generate_wandb(fpath, ftarget, model_path_fold_first)
        dconfig["model_path_fold_first"] = model_path_fold_first
        self.write_config(dataset_name, dconfig, CONFIG_FILE)
        self.generate_sweep(wandb_key, pred_dir, launch_file, ftarget, generate_all)
```

Example:

```
df = wandb_api.get_best_run(dataset_name="assist2015",model_name="dkt")
wandb_api.extract_best_models(df, dataset_name, model_name,
                              fpath="../examples/seedwandb/predict.yaml",wandb_key=wandb_
```

(continues on next page)

(continued from previous page)

```
↪key)
```

- sh [launch_file] [parameter]

After extracting the best model, we can get the lauch file for evaluation automatically, the default filename is "start_pred.sh". Then we can start sweep for prediction.

```
sh [launch_file] > [Directed log] 2>&1

    - [launch_file]: required, the user submits the script of sweep to wandbs, and␣
↪directs the execution output to [directed log])
    - [Directed log]: Required, execute the sweep in the log
```

Example:

```
sh start_predict.sh > pred.log 2>&1
(You need to define the log file. )
```

- sh run_all.sh [parameter]

Example:

```
WANDB_API_KEY=xxx sh run_all.sh pred.log 0 1 assist2009 dkt 0 nips2022-assist2009
```

- Start Agents

```
sh start_sweep_0_1.sh
```

There are only 5 sweeps to be run without any parameter tuning in this stage, with each sweep corresponding to the evaluation of each fold of the training data. Finally, you can export the evaluation results externally or call the wandb API for statistical 5- folds results, and calculate the mean and standard deviation of each metric, i.e., ***mean*** $\pm$ ***standard deviation***

If you want to add new models or datasets into pyKT, you can follow *Contribute*.

CHAPTER

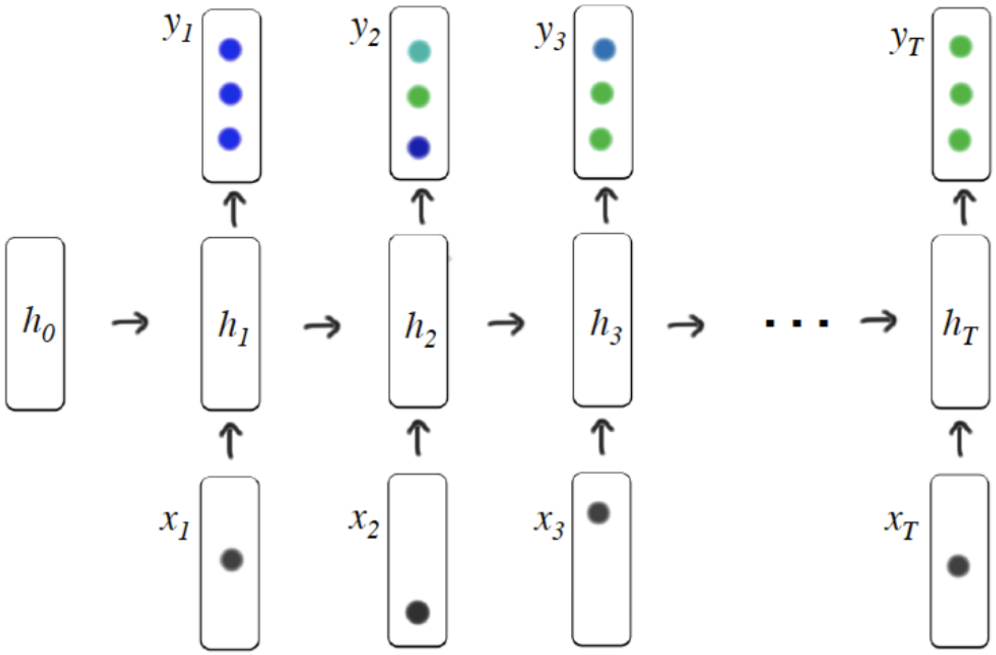
TWO

MODELS

Since the deep learning based KT models can be categorized into deep sequential models, memory augmented models, adversarial based models, graph based models and attention based models in our work, we mainly develop the DLKT models by these four categories in pyKT.

2.1 DKT

DKT is the first model that uses Recurrent Neural Networks (RNNs) to solve Knowledge Tracing.



Piech, Chris, et al. "Deep knowledge tracing." Advances in neural information processing systems 28 (2015).

2.2 DKT+

DKT+ introduces regularization terms that correspond to reconstruction and waviness to the loss function of the original DKT model to enhance the consistency in KT prediction.

Yeung, Chun-Kit, and Dit-Yan Yeung. "Addressing two problems in deep knowledge tracing via prediction-consistent regularization." Proceedings of the Fifth Annual ACM Conference on Learning at Scale. 2018.

2.3 DKT-Forget

DKT-Forget explores the deep knowledge tracing model by considering the forgetting behavior via incorporate multiple forgetting information.

(a) Architecture for deep knowledge tracing.

(b) Architecture for proposed model.

Nagatani, Koki, et al. "Augmenting knowledge tracing by considering forgetting behavior." The world wide web conference. 2019.

2.4 KQN

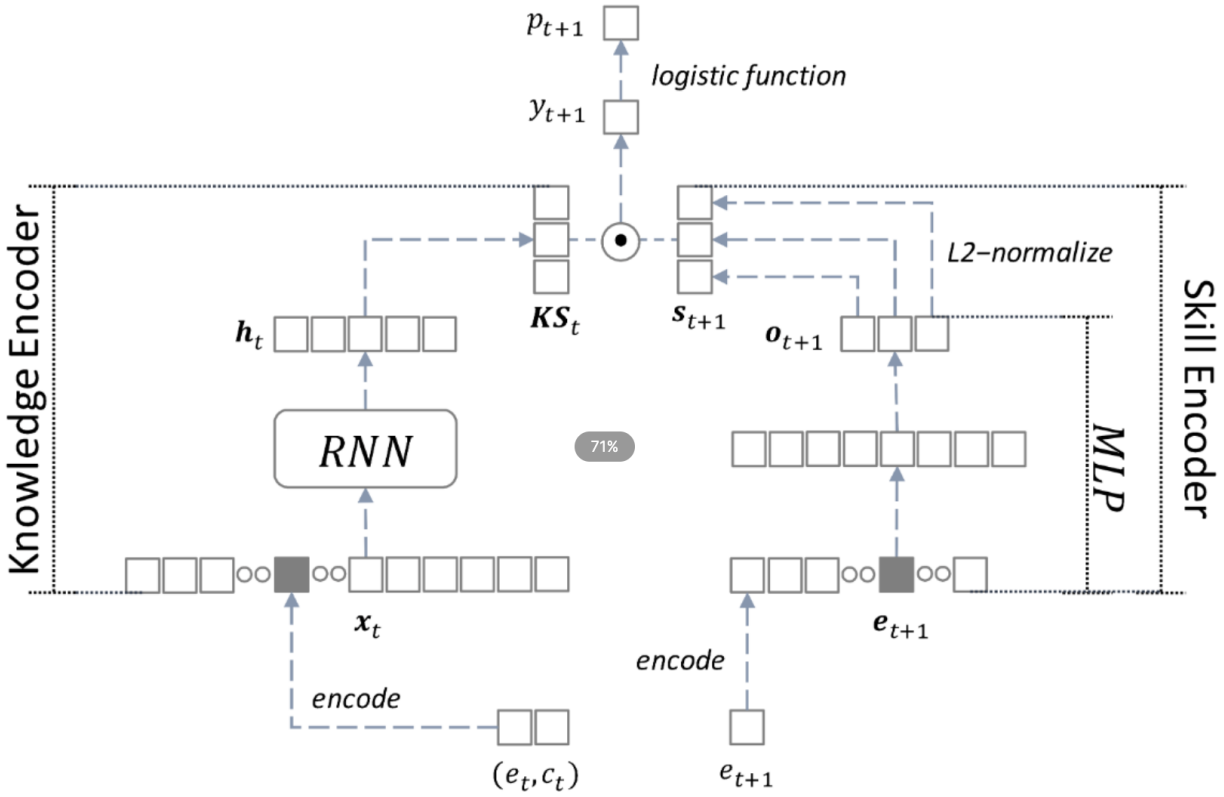
KQN uses neural networks to encode student learning activities into knowledge state and skill vectors, and calculate the relations between the interactions via dot product.

Lee, Jinseok, and Dit-Yan Yeung. "Knowledge query network for knowledge tracing: How knowledge interacts with skills." Proceedings of the 9th international conference on learning analytics & Knowledge. 2019.

2.5 DKVMN

Dynamic key-value memory networks (DKVMN) exploit the relationships between latent KCs which are stored in a static memory matrix *key* and predict the knowledge mastery level of a student directly based on a dynamic memory matrix *value*.

Zhang, Jiani, et al. "Dynamic key-value memory networks for knowledge tracing." Proceedings of the 26th international conference on World Wide Web. 2017.



2.6 ATKT

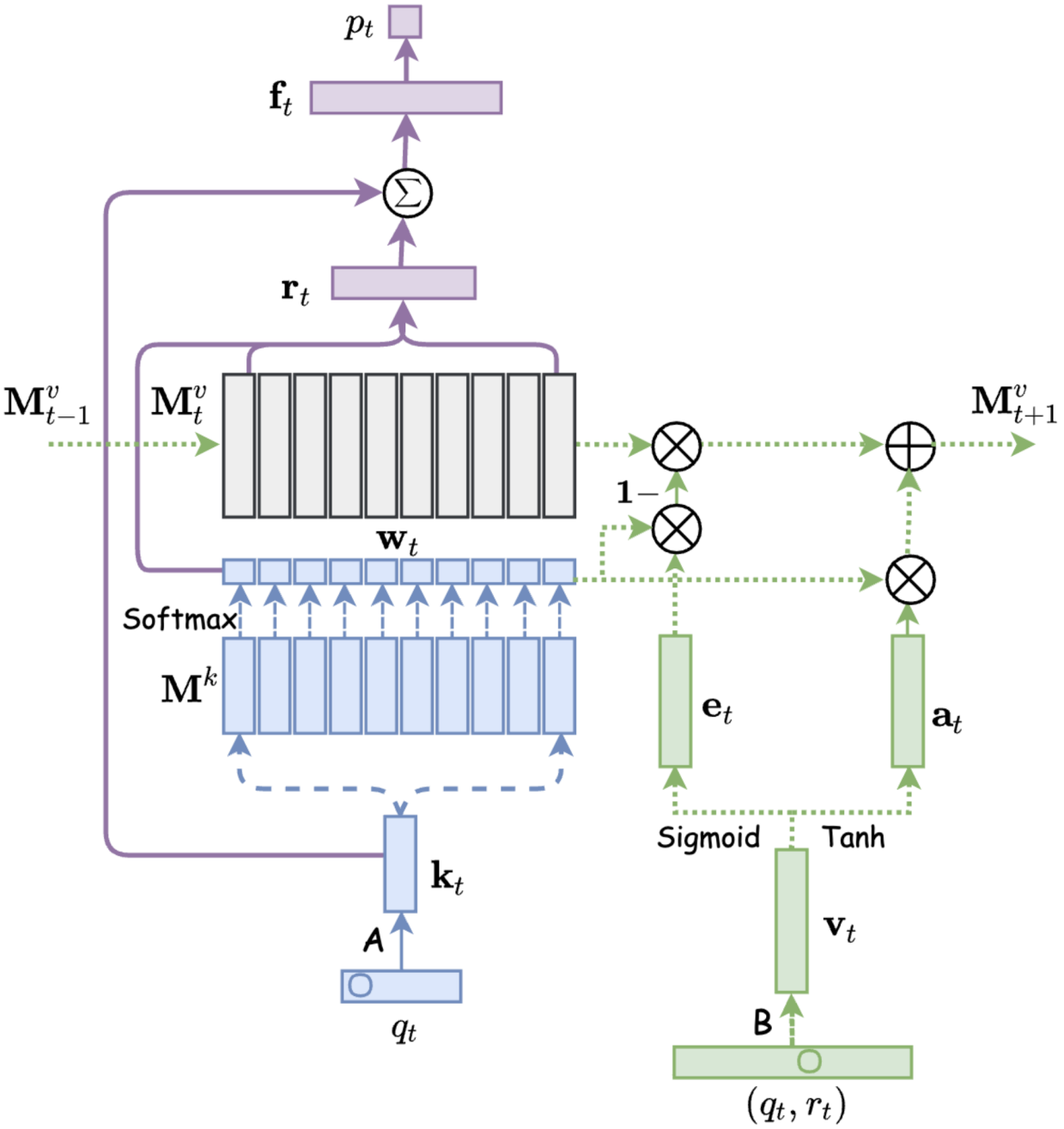
Adversarial training (AT) based KT method (ATKT) is an attention based LSTM model which apply the adversarial perturbations into the original student interaction sequence to reduce the the risk of DLKT overfitting and limited generalization problem.

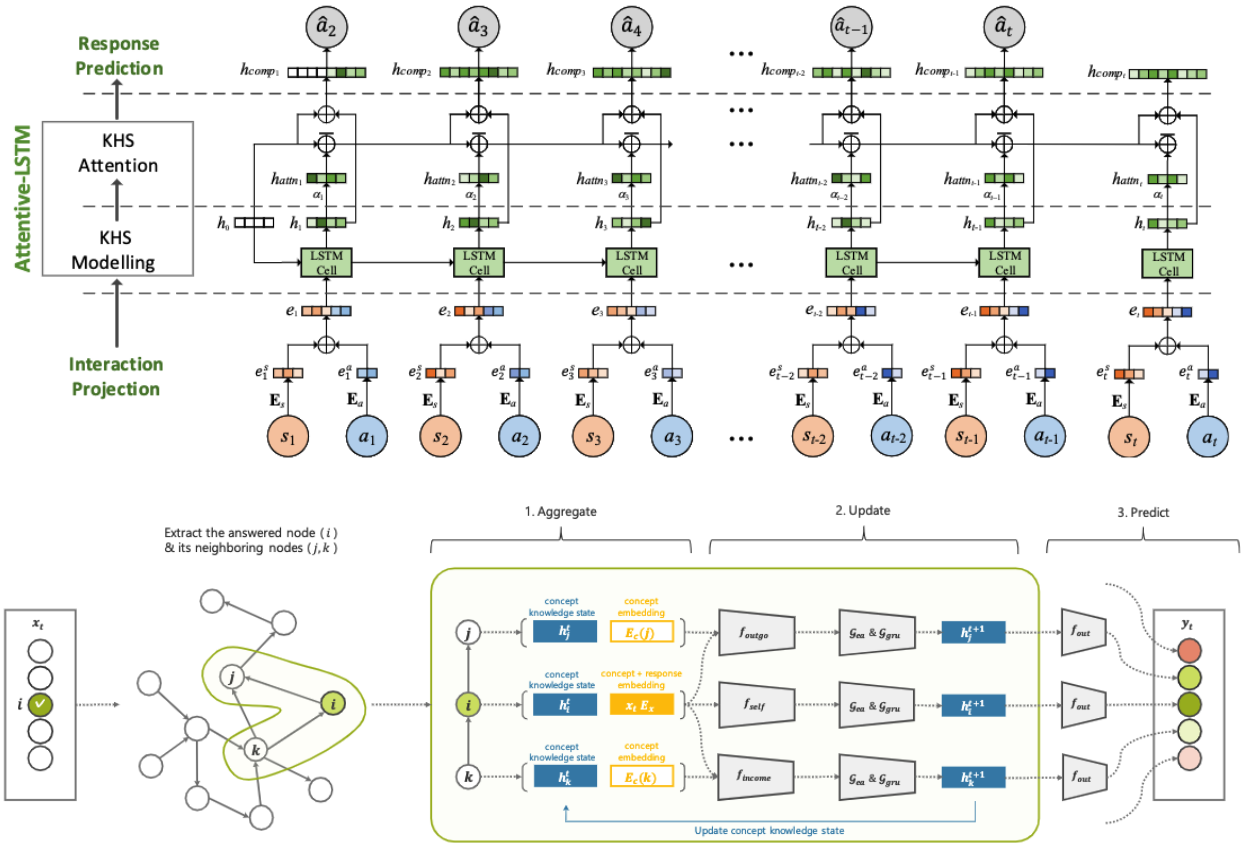
Guo, Xiaopeng, et al. "Enhancing Knowledge Tracing via Adversarial Training." Proceedings of the 29th ACM International Conference on Multimedia. 2021.

2.7 GKT

Graph-based Knowledge Tracing (GKT) is a GNN-based knowledge tracing method that use a graph to model the relations between knowledge concepts to reformulate the KT task as a time-series node-level classification problem.

Nakagawa, Hiromi, Yusuke Iwasawa, and Yutaka Matsuo. "Graph-based knowledge tracing: modeling student proficiency using graph neural network." 2019 IEEE/WIC/ACM International Conference On Web Intelligence (WI). IEEE, 2019.


p_t
$\mathbf{f}_t$
Σ
$\mathbf{r}_t$
$\mathbf{M}^v_{t-1}$
$\mathbf{M}^v_t$
$\mathbf{M}^v_{t+1}$
$\mathbf{w}_t$
1 −
Softmax
$\mathbf{M}^k$
$\mathbf{e}_t$
$\mathbf{a}_t$
Sigmoid
Tanh
$\mathbf{k}_t$
$\mathbf{v}_t$
A
B
q_t
(q_t, r_t)



2.8 SAKT

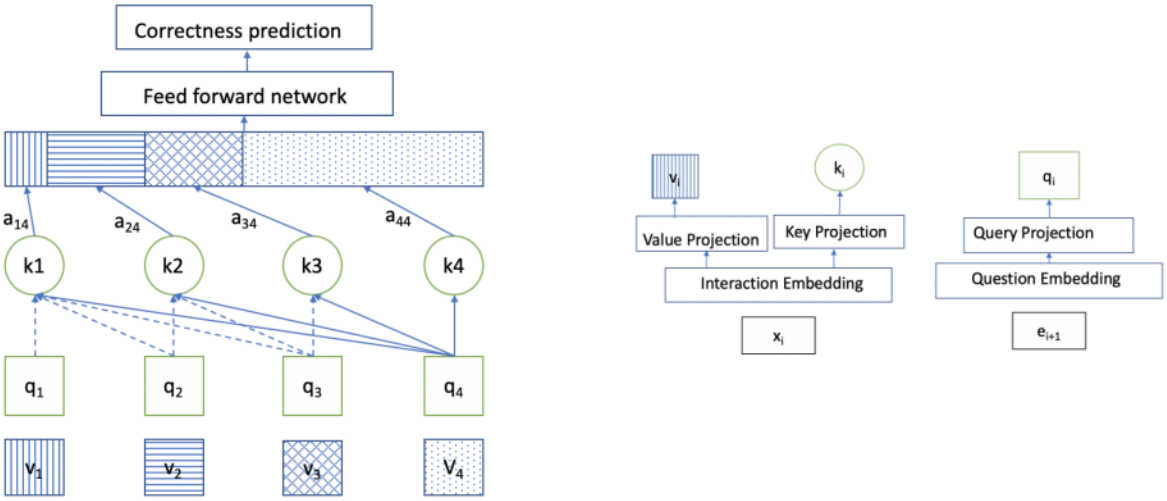
Self Attentive Knowledge Tracing (SAKT) use self-attention network to capture the relevance between the KCs and the students' historical interactions.

Pandey, Shalini, and George Karypis. "A self-attentive model for knowledge tracing." arXiv preprint arXiv:1907.06837 (2019).

2.9 SAINT

Separated Self-AttentIve Neural Knowledge Tracing(SAINT) is a typical Transformer based structure which embeds the exercises in encoder and predict the responses in decoder.

Choi, Youngduck, et al. "Towards an appropriate query, key, and value computation for knowledge tracing." Proceedings of the Seventh ACM Conference on Learning@ Scale. 2020.



2.10 AKT

Attentive knowledge tracing (AKT) introduce a rasch model to

regularize the KC and question embeddings to discriminate the questions on the same KC, and modeling the exercise representations and the students' historical interactdion embeddings via three self-attention based modules.

Ghosh, Aritra, Neil Heffernan, and Andrew S. Lan. "Context-aware attentive knowledge tracing." Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining. 2020.

2.11 SKVMN

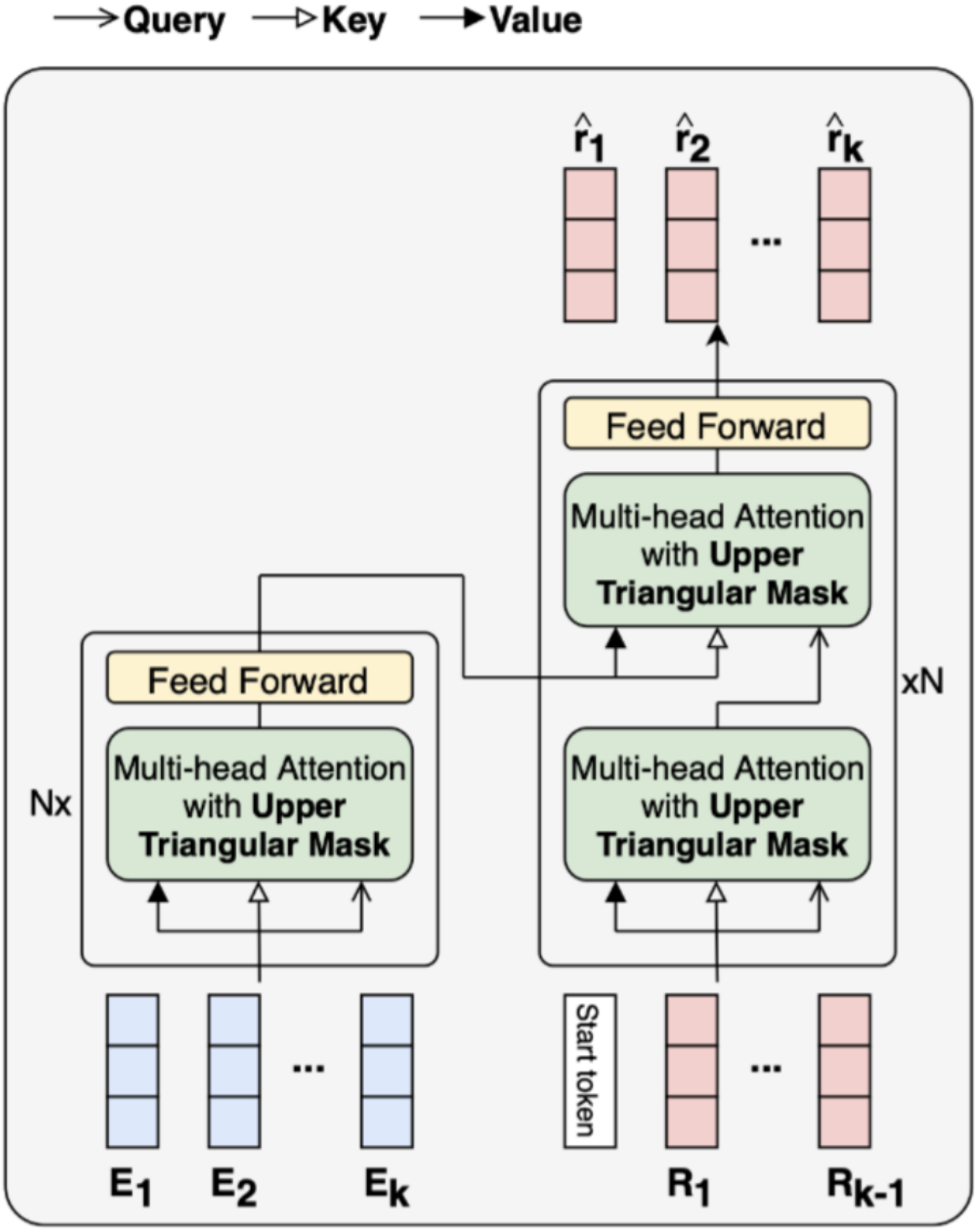
This model unifies the strengths of recurrent modeling capacity and the capability of memory networks to model the students' learning precocesses.

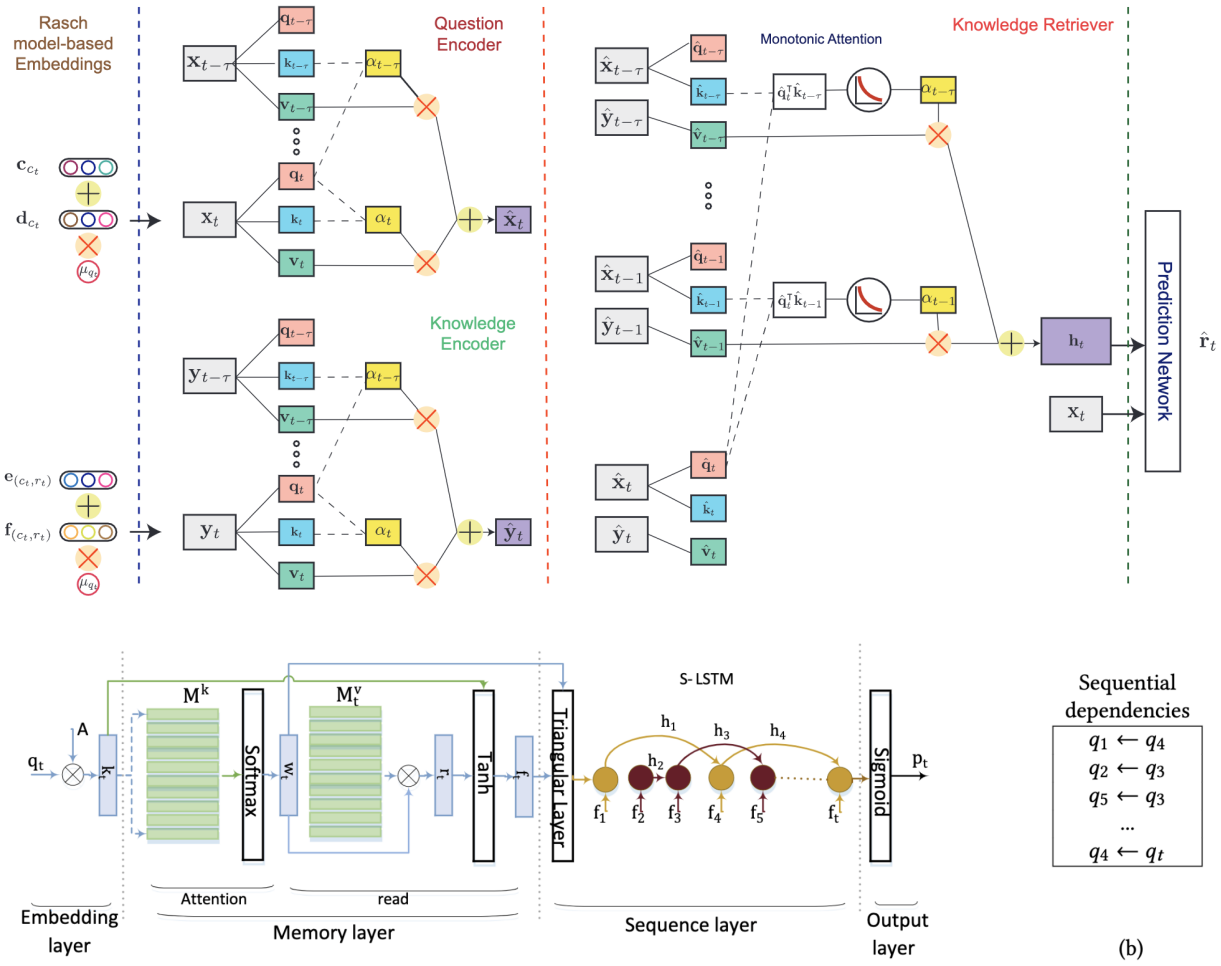
Abdelrahman, Ghodai, and Qing Wang. "Knowledge tracing with sequential key-value memory networks." Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2019.

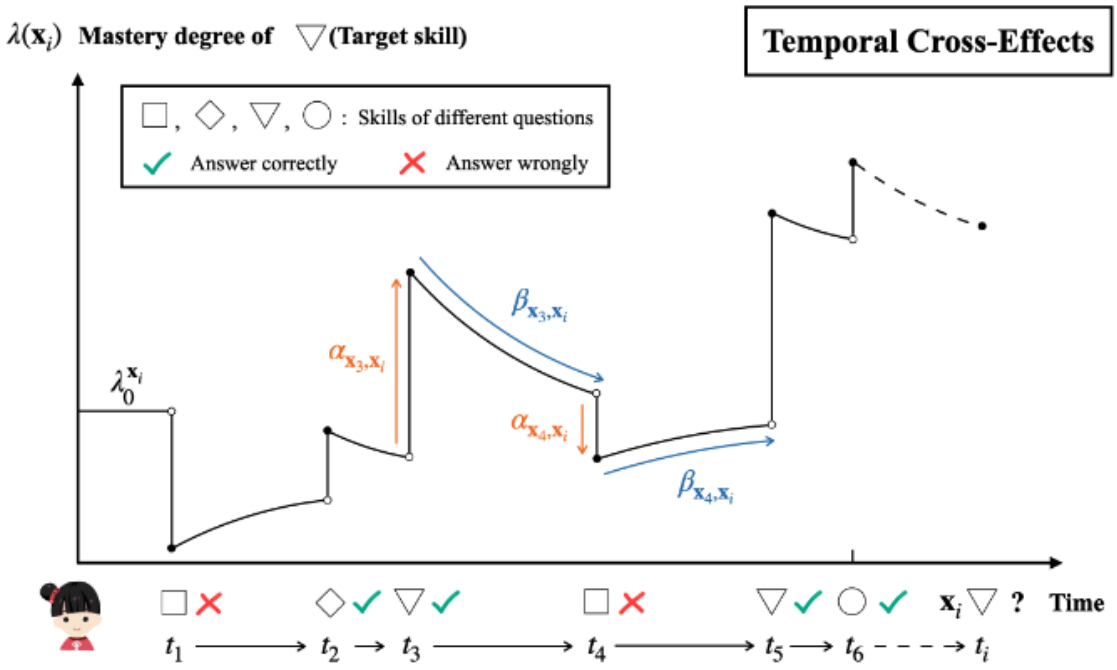
2.12 HawkesKT

HawkesKT is the first to introduce Hawkes process to model temporal cross effects in KT.

Wang, Chenyang, et al. "Temporal cross-effects in knowledge tracing." Proceedings of the 14th ACM International Conference on Web Search and Data Mining. 2021.

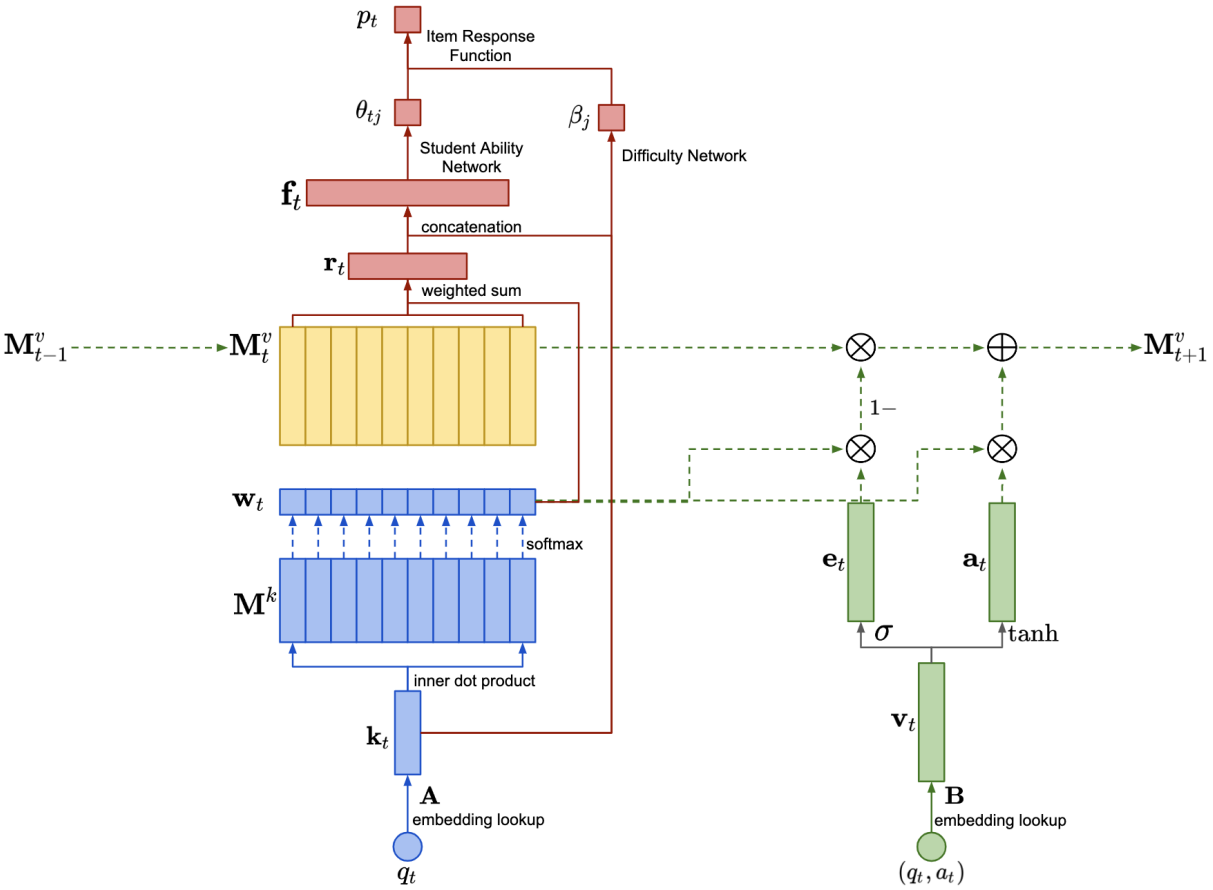

Query
Key
Value
$\hat{r}_1$
$\hat{r}_2$
$\hat{r}_k$
Feed Forward
Multi-head Attention with Upper Triangular Mask
Feed Forward
Multi-head Attention with Upper Triangular Mask
Nx
xN
Start token
E_1
E_2
E_k
R_1
R_{k-1}


Rasch model-based Embeddings
Question Encoder
Knowledge Encoder
Monotonic Attention
Knowledge Retriever
Prediction Network
Softmax
Tanh
Triangular layer
S- LSTM
Sigmoid
Embedding layer
Attention
read
Memory layer
Sequence layer
Output layer
Sequential dependencies
$q_1 \leftarrow q_4$
$q_2 \leftarrow q_3$
$q_5 \leftarrow q_3$
...
$q_4 \leftarrow q_t$
(b)


$\lambda(\mathbf{x}_i)$ Mastery degree of (Target skill)
Temporal Cross-Effects
: Skills of different questions
Answer correctly
Answer wrongly
$\lambda_0^{\mathbf{x}_i}$
$\alpha_{\mathbf{x}_3,\mathbf{x}_i}$
$\beta_{\mathbf{x}_3,\mathbf{x}_i}$
$\alpha_{\mathbf{x}_4,\mathbf{x}_i}$
$\beta_{\mathbf{x}_4,\mathbf{x}_i}$
$\mathbf{x}_i$?
Time
t_1 t_2 t_3 t_4 t_5 t_6 t_i

2.13 Deep-IRT

Deep-IRT is a synthesis of the item response theory (IRT) model and a knowledge tracing model that is based on the deep neural network architecture called dynamic key-value memory network (DKVMN) to make deep learning based knowledge tracing explainable.

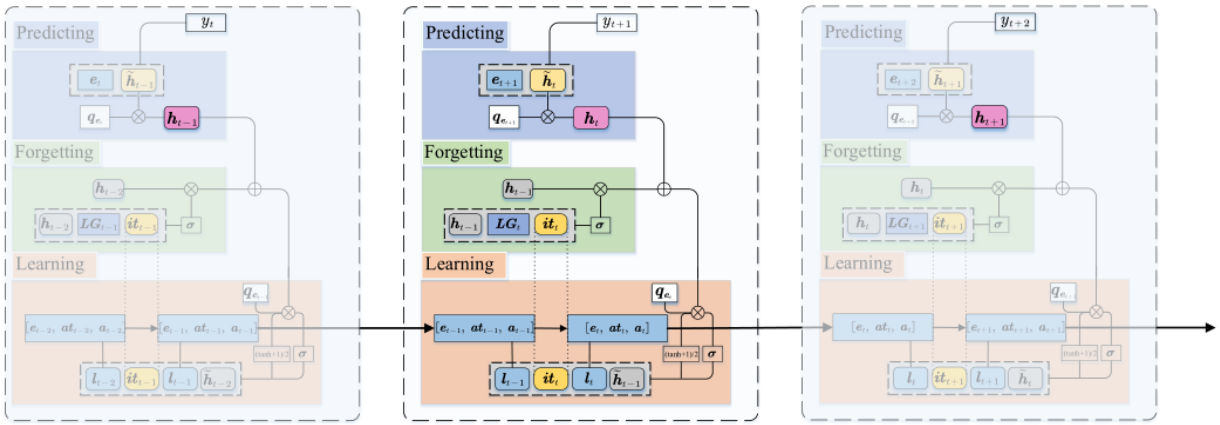


Yeung, Chun-Kit. "Deep-IRT: Make deep learning based knowledge tracing explainable using item response theory." arXiv preprint arXiv:1904.11738 (2019).

2.14 LPKT

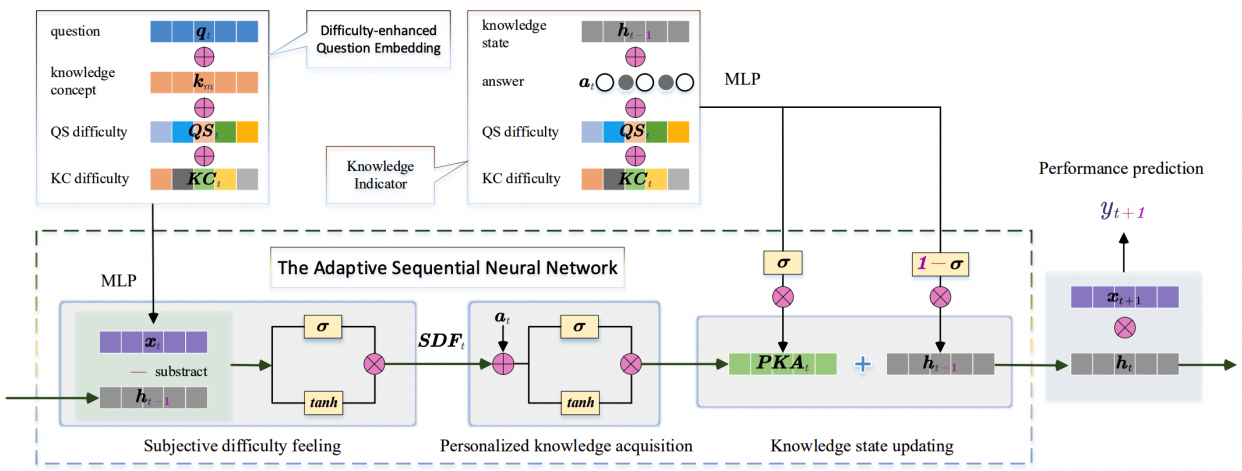
Learning Processconsistent Knowledge Tracing(LPKT) monitors students' knowledge state by directly modeling their learning process.

Shen, Shuanghong, et al. "Learning process-consistent knowledge tracing." Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021.



2.15 DIMKT

The DIfficulty Matching Knowledge Tracing (DIMKT) model explicitly incorporate the difficulty level into the question representation and establish the relation between students' knowledge state and the question difficulty level during the practice process.

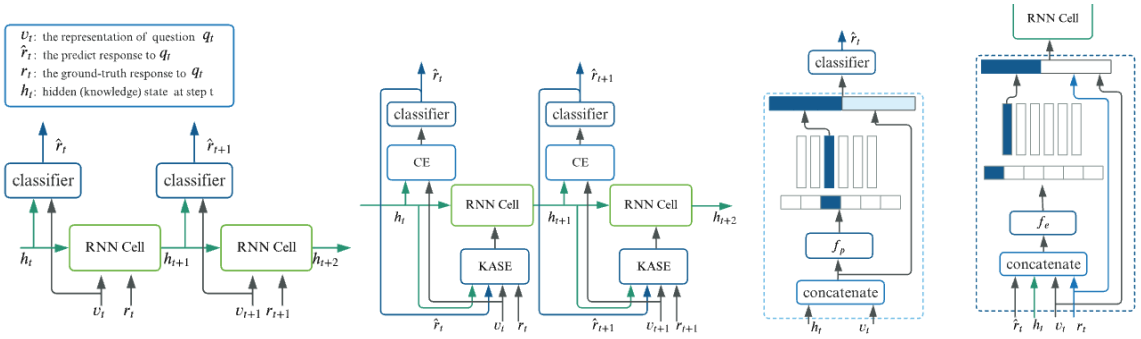


Shen, Shuanghong, et al. "Assessing Student's Dynamic Knowledge State by Exploring the Question Difficulty Effect." Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2022.

2.16 IEKT

Individual Estimation Knowledge Tracing (IEKT) estimates the students' cognition of the question before response prediction and assesses their knowledge acquisition sensitivity on the questions before updating the knowledge state.

Long, Ting, et al. "Tracing knowledge state with individual cognition and acquisition estimation." Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2021.



(a) The previous RNN-based works. (b) The framework of IEKT. (c) The CE modules. (d) The KASE module.

2.17 qDKT

qDKT(base) is a model same as DKT, but use the question ID as the input.

Sonkar, Shashank, et al. "qdkt: Question-centric deep knowledge tracing." arXiv preprint arXiv:2005.12442 (2020).

2.18 AT-DKT

AT-DKT improve the prediction performance of the original deep knowledge tracing model with two auxiliary learning tasks including question tagging prediction task and individualized prior knowledge prediction task.

Liu, Zitao, et al. "Enhancing deep knowledge tracing with auxiliary tasks." Proceedings of the ACM Web Conference. 2023.

2.19 simpleKT

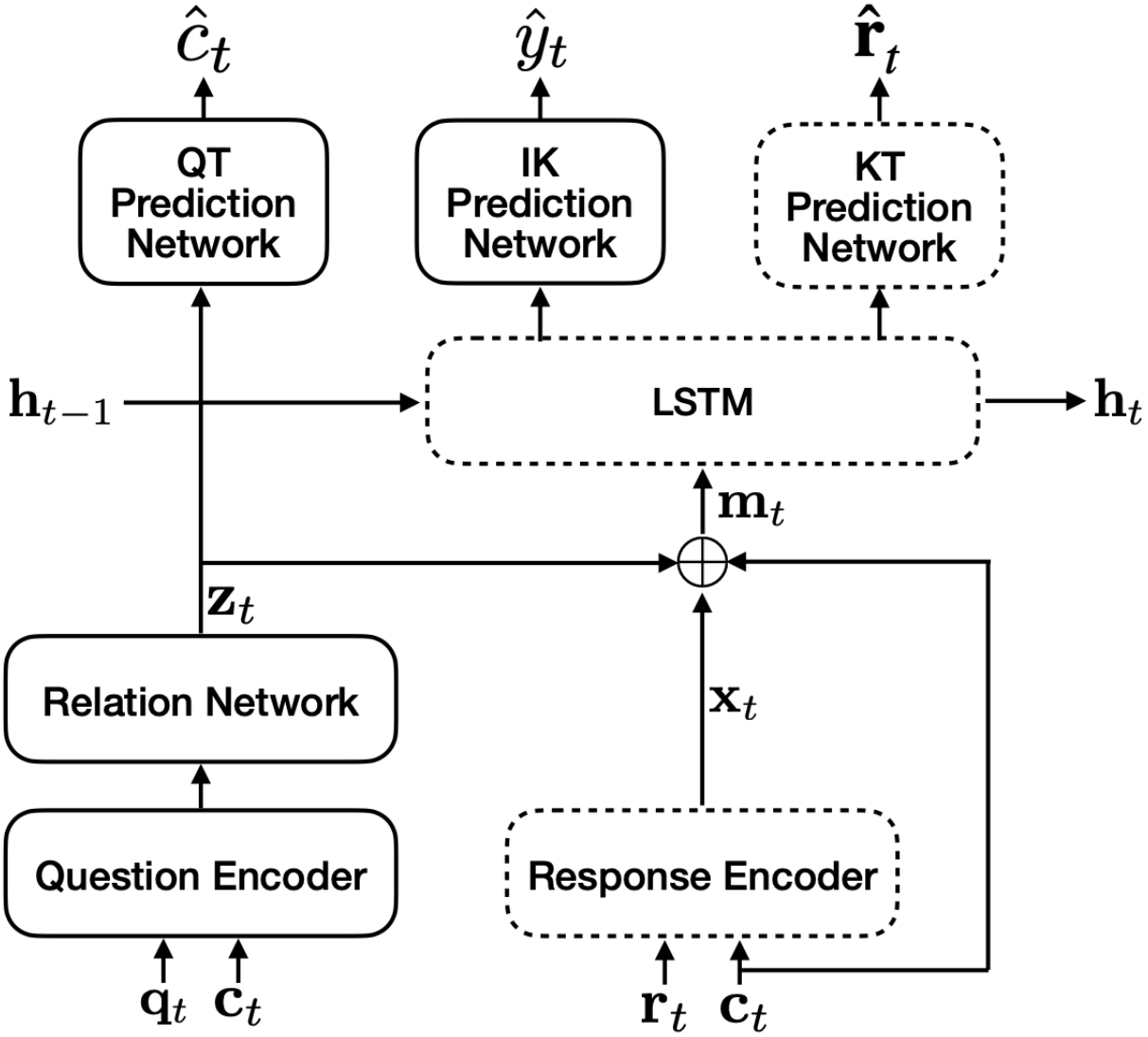
simpleKT is a strong but simple baseline method to deal with the KT task by modeling question-specific variations based on Rasch model and use the ordinary dot-product attention function to extract the time-aware information embedded in the student learning interactions.

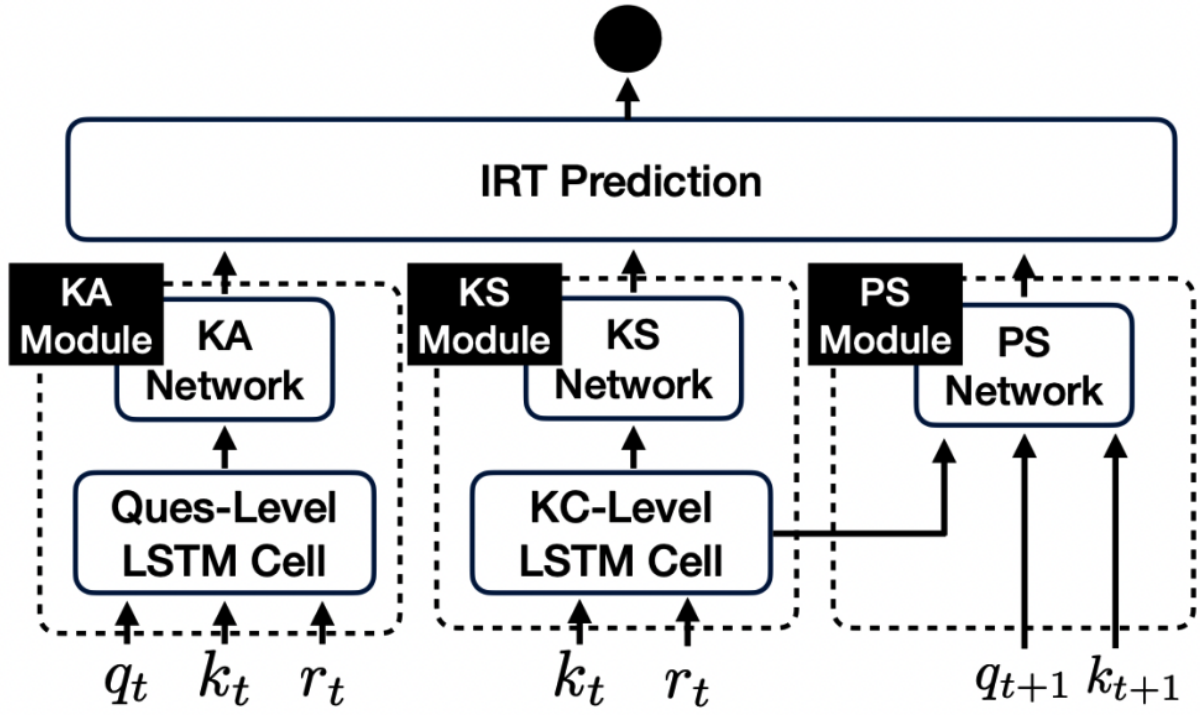
Liu, Zitao, et al. "simpleKT: A Simple But Tough-to-Beat Baseline for Knowledge Tracing." The Eleventh International Conference on Learning Representations. 2022.

2.20 QIKT

QIKT is a question-centric interpretable KT model that estimates students' knowledge state variations at a fine-grained level with question-sensitive cognitive representations that are jointly learned from a question-centric knowledge acquisition module and a question-centric problem solving module.

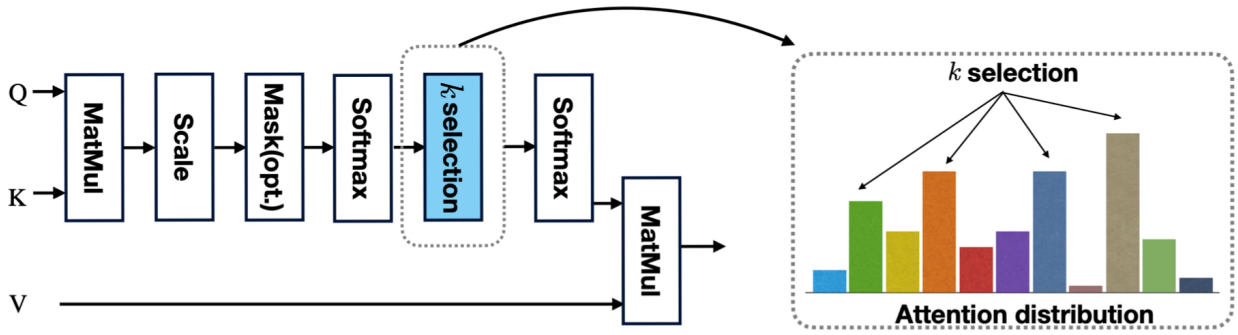
Chen, Jiahao, et al. "Improving interpretability of deep sequential knowledge tracing models with question-centric cognitive representations." The 37th AAAI Conference on Artificial Intelligence. 2023.


$\hat{c}_t$
$\hat{y}_t$
$\hat{\mathbf{r}}_t$
QT Prediction Network
IK Prediction Network
KT Prediction Network
$\mathbf{h}_{t-1}$
LSTM
$\mathbf{h}_t$
$\mathbf{m}_t$
$\mathbf{z}_t$
$\mathbf{x}_t$
Relation Network
Question Encoder
Response Encoder
$\mathbf{q}_t$ $\mathbf{c}_t$
$\mathbf{r}_t$ $\mathbf{c}_t$



2.21 sparseKT-soft/topK

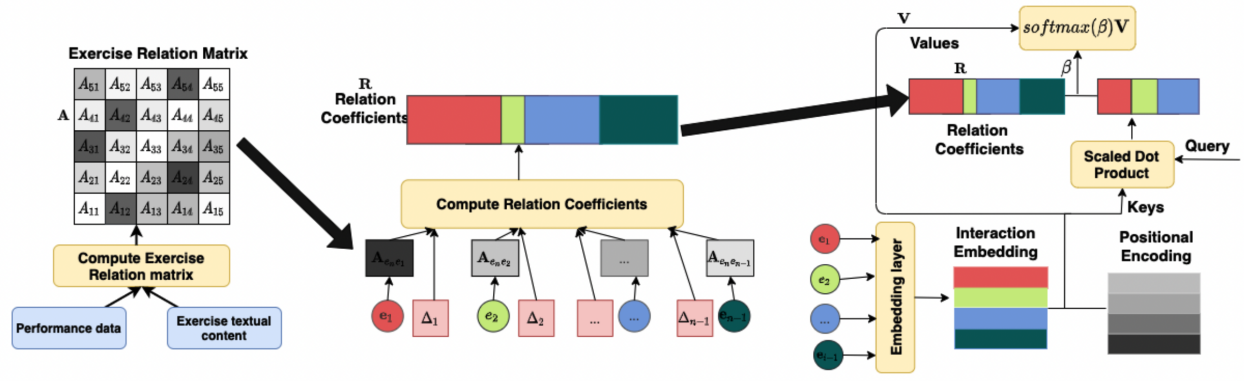
sparseKT incorporate a k-selection module to only pick items with the highest attention scores including two sparsification heuristics: (1) soft-thresholding sparse attention (sparseKT-soft) and (2) top- sparse attention (sparseKT-topK).



Shuyan Huang, et al. "Towards Robust Knowledge Tracing Models via k-Sparse Attention." Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2023.

2.22 RKT

RKT contains a relation-aware self-attention layer that incorporates the contextual information including both the exercise relation information through their textual content as well as student performance data and the forget behavior information through modeling an exponentially decaying kernel function.



Pandey, Shalini, and Jaideep Srivastava. "RKT: relation-aware self-attention for knowledge tracing." Proceedings of the 29th ACM International Conference on Information & Knowledge Management. 2020.

2.23 FoLiBiKT

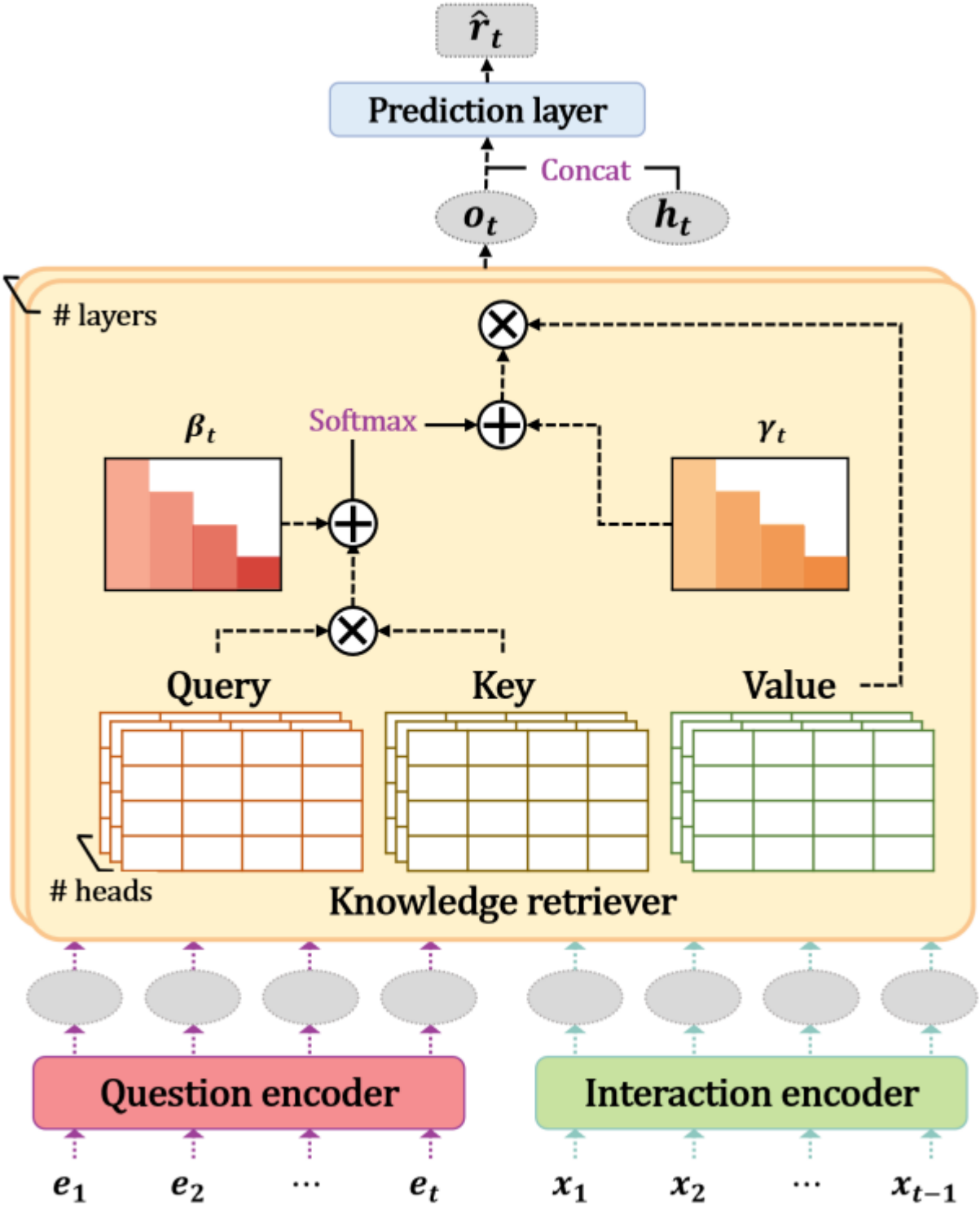
FoLiBi (Forgetting-aware Linear Bias) is a simple yet effective solution that introduces a linear bias term to explicitly model learners' forgetting behavior, compensating for the neglect of forgetting effects in existing attention-based Knowledge Tracing modelsWe reproduced FoLiBi with AKT, namely FoLiBiKT.

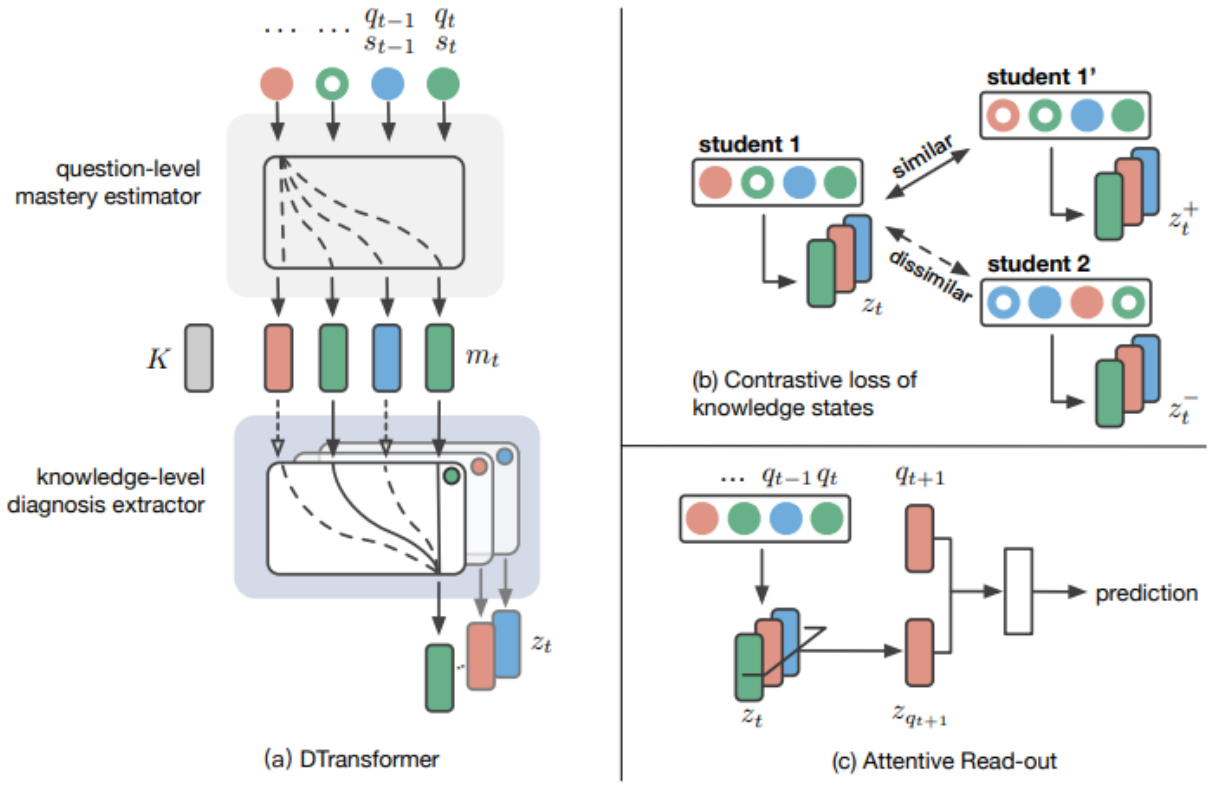
Im, Yoonjin, et al. "Forgetting-aware Linear Bias for Attentive Knowledge Tracing." Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. 2023.

2.24 Dtransformer

The Diagnostic Transformer (DTransformer) integrates question-level mastery with knowledge-level diagnosis using Temporal and Cumulative Attention (TCA) and multi-head attention for dynamic knowledge tracing. Moreover, a contrastive learning-based training algorithm enhances the stability of knowledge state diagnosis.

Yin, Yu, et al. "Tracing Knowledge Instead of Patterns: Stable Knowledge Tracing with Diagnostic Transformer." Proceedings of the ACM Web Conference. 2023.


$\hat{r}_t$
Prediction layer
Concat
o_t
h_t
layers
β_t
Softmax
γ_t
Query
Key
Value
heads
Knowledge retriever
Question encoder
Interaction encoder
e_1
e_2
$\cdots$
e_t
x_1
x_2
$\cdots$
x_{t-1}



(a) DTransformer

(b) Contrastive loss of knowledge states

(c) Attentive Read-out

CHAPTER

THREE

DATASETS

We mainly select the following datasets in the pyKT at presents:

Dataset	question ID	skill ID	answering results	answering duration	answer submission time
Statics20 11	✓		✓		✓
ASSISTments2009	✓	✓	✓		
ASSISTments2012	✓	✓	✓	✓	✓
ASSISTments2015		✓	✓		
ASSISTments2017	✓	✓	✓	✓	✓
Algebra20 05	✓	✓	✓		✓
Bridge200 6	✓	✓	✓		✓
Ednet	✓	✓	✓	✓	✓
NIPS34	✓	✓	✓		✓
POJ	✓		✓		✓

3.1 Statics2011

This dataset is collected from an engineering statics course taught at the Carnegie Mellon University during Fall 2011. In this dataset, a unique question is constructed by concatenating the problem name and step name and the dataset has 194,947 interactions, 333 students, 1,224 questions.

https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=507

3.2 ASSISTments2009

This dataset is made up of math exercises, collected from the free online tutoring ASSISTments platform in the school year 2009-2010. The dataset consists of 346,860 interactions, 4,217 students, and 26,688 questions and is widely used and has been the standard benchmark for KT methods over the last decade.

https://sites.google.com/site/assistmentsdata/home/2009-2010-assistment-data/skill-builder-data-2009-2010

3.3 ASSISTments2012

This is the ASSISTments data for the school year 2012~2013 with affect predictions. The dataset consists of 2,541,201 interactions, 27,066 students, and 45,716 questions.

https://sites.google.com/site/assistmentsdata/datasets/2012-13-school-data-with-affect

3.4 ASSISTments2015

Similar to ASSISTments2009, this dataset is collected from the ASSISTments platform in the year of 2015. It includes 708,631 interactions on 100 distinct KCs from 19,917 students. This dataset has the largest number of students among the other ASSISTments datasets.

https://sites.google.com/site/assistmentsdata/datasets/2015-assistments-skill-builder-data

3.5 ASSISTments2017

This dataset is from the 2017 data mining competition. It consists of 942,816 interactions, 686 students, and 102 questions.

https://sites.google.com/view/assistmentsdatamining/dataset?authuser=0

3.6 Algebra2005

This dataset is from the KDD Cup 2010 EDM Challenge that contains 13-14 year old students' responses to Algebra questions. It contains detailed step-level student responses. The unique question construction is similar to the process used in Statics2011, which ends up with 809,694 interactions, 574 students, 210,710 questions and 112 KCs.

https://pslcdatashop.web.cmu.edu/KDDCup/

3.7 Bridge2006

This dataset is also from the KDD Cup 2010 EDM Challenge and the unique question construction is similar to the process used in Statics2011. There are 3,679,199 interactions, 1,146 students, 207,856 questions and 493 KCs in the dataset.

https://pslcdatashop.web.cmu.edu/KDDCup/

3.8 Ednet

The large-scale hierarchical student activity data set collected by Santa (an artificial intelligence guidance system) contains 131317236 interactive information of 784309 students, which is the largest public interactive education system data set released so far.

https://github.com/riiid/ednet

3.9 NIPS34

This dataset is from the Tasks 3 & 4 at the NeurIPS 2020 Education Challenge. It contains students' answers to multiple-choice diagnostic math questions and is collected from the Eedi platform. For each question, we choose to use the leaf nodes from the subject tree as its KCs, which ends up with 1,382,727 interactions, 948 questions, and 57 KCs.

https://eedi.com/projects/neurips-education-challenge

3.10 POJ

This dataset consists of programming exercises and is collected from Peking coding practice online platform. The dataset is originally scraped by Pandey and Srivastava. In total, it has 996,240 interactions, 22,916 students, and 2,750 questions.

https://drive.google.com/drive/folders/1LRljqWfODwTYRMPw6wEJ_mMt1KZ4xBDk

CHAPTER

FOUR

HOW TO CONTRIBUTE TO PYKT?

pyKT is still under development. More KT models and datasets will be added, and we always welcome contributions to make pyKT better.

4.1 Guidance

Thank you for your interest in contributing to pyKT! You can make the following contributions to pyKT:

1. Bug-fix for an outstanding issue.
2. Add new datasets.
3. New model implementations.

4.1.1 Install for Development

1Fork the pyKT by clicking the "Fork" button.

2Clone the repository you cloned, and switch to the dev branch (Notice: Do not work on the main branch).

```
git clone https://github.com/{your github name}/pykt-toolkit
cd pykt-toolkit
git checkout dev
```

3Editable Installation

You can use the following command to install the pyKT library.

```
pip install -e .
```

In this way, every change made to the `pykt` directory will be effective immediately. The package does not require another installation again.

4.1.2 Push Your Codes to pyKT (Pull Requests)

After implementing the new models or fixing bugs, you can push your codes to the dev branch in your repository. Then, you can use the merge request feature to merge your codes to pyKT's **main** branch.

The main branch is **not be allowed** to push codes (the push submission will be failed). And we will refuse the Pull Request from other branches to the main branch except for dev branch.

4.2 Add Your Datasets

In this section, we will use the `ASSISTments2015` dataset as an example to show the adding dataset procedure. Here we simplify the `ASSISTments2015` into `assist2015` as the dataset name, you can replace `assist2015` in your own dataset.

4.2.1 Create Data Files

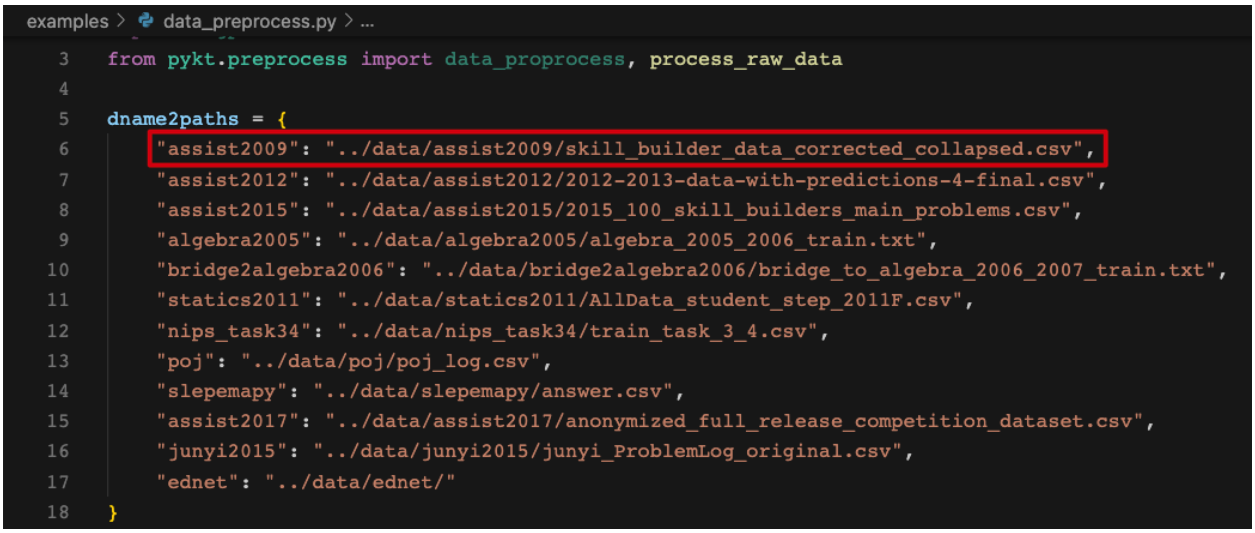
1Please create a new dataset folder in the `data` directory with dataset name.

```
mkdir -p ./data/assist2015
```

2You should add the raw files of the new dataset into the named directory, like the file structure of `assist2015`:

```
$tree data/assist2015/
├── 2015_100_skill_builders_main_problems.csv
```

3You need to provide the data path of new dataset to `dname2paths` of `examples/data_preprocess.py`.



4.2.2 Data Preprocess File

1Create the processing file `assist2015_preprocess.py` under the `pykt/preprocess` directory. The data preprocessing are suggested to follow the [Data Preprocess Guidelines](#Data Preprocess Guidelines). The main codes of the data preprocessing of `assist2015` are as follows:

```
import pandas as pd
from pykt.utils import write_txt, change2timestamp, replace_text

def read_data_from_csv(read_file, write_file):
    # load the original data
    df = pd.read_table(read_file, encoding = "utf-8", dtype=str, low_memory=False)
    df["Problem Name"] = df["Problem Name"].apply(replace_text)
    df["Step Name"] = df["Step Name"].apply(replace_text)
    df["Questions"] = df.apply(lambda x:f"{x['Problem Name']}----{x['Step Name']}",
↪axis=1)


    df["index"] = range(df.shape[0])
    df = df.dropna(subset=["Anon Student Id", "Questions", "KC(Default)", "First␣
↪Transaction Time", "Correct First Attempt"])
    df = df[df["Correct First Attempt"].isin([str(0),str(1)])]#keep the interaction␣
↪which response in [0,1]
    df = df[["index", "Anon Student Id", "Questions", "KC(Default)", "First Transaction␣
↪Time", "Correct First Attempt"]]
    df["KC(Default)"] = df["KC(Default)"].apply(replace_text)

    data = []
    ui_df = df.groupby(['Anon Student Id'], sort=False)

    for ui in ui_df:
        u, curdf = ui[0], ui[1]
        curdf.loc[:, "First Transaction Time"] = curdf.loc[:, "First Transaction Time"].
↪apply(lambda t: change2timestamp(t))
        curdf = curdf.sort_values(by=["First Transaction Time", "index"])
        curdf["First Transaction Time"] = curdf["First Transaction Time"].astype(str)

        seq_skills = [x.replace("~~", "_") for x in curdf["KC(Default)"].values]
        seq_ans = curdf["Correct First Attempt"].values
        seq_start_time = curdf["First Transaction Time"].values
        seq_problems = curdf["Questions"].values
        seq_len = len(seq_ans)
        seq_use_time = ["NA"]

        data.append(
            [[u, str(seq_len)], seq_problems, seq_skills, seq_ans, seq_start_time, seq_
↪use_time])

    write_txt(write_file, data)
```

The entire codes can be seen in `pykt/preprocess/algebra2005_preprocess.py`.

2Import the preprocess file to `pykt/preprocess/data_proprocess.py`.

pykt > preprocess > data_proprocess.py > ...

TongZhuzzz, 2 weeks ago | 3 authors (You and others)

```python
import os, sys

def process_raw_data(dataset_name,dname2paths):
    readf = dname2paths[dataset_name]
    dname = "/".join(readf.split("/")[0:-1])
    writef = os.path.join(dname, "data.txt")
    print(f"Start preprocessing data: {dataset_name}")
    if dataset_name == "assist2009":
        from .assist2009_preprocess import read_data_from_csv
    elif dataset_name == "assist2012":
        from .assist2012_preprocess import read_data_from_csv
    elif dataset_name == "assist2015":
        from .assist2015_preprocess import read_data_from_csv
    elif dataset_name == "algebra2005":
        from .algebra2005_preprocess import read_data_from_csv
```

4.2.3 Data Preprocess Guidelines

Field Extraction

For each datset, we mainly extract 6 fields for model training: user ID, question ID (name), skill ID (name), answering results, answer submission time, and answering duration(if the field does not exist in the dataset, it is represented by "NA").

Data Filtering

The interaction with lacks one of the five fields in user ID, question ID (name), skill ID (name), answering results, answer submission time will be filtered out.

Data Ordering

A student's interaction sequence is order according to the answer submission time. The order will be kept consistent with the original relative position for different interactions with the same submission time.

Character Processing

- **Field concatenation:** Use `----` as the connecting symbol. For example, Algebra2005 needs to concatenate `Problem Name` and `Step Name` as the final problem name.
- **Character replacement:** If there is an underline _ in the question and skill of original data, replace it with `####`. If there is a comma `,` in the question and skill of original data, replace it with `@@@@`.
- **Multi-skill separator:** If there are multiple skills in a question, we separate the skills with an underline _.
- **Time format:** The answer submission time is a millisecond (ms) timestamp, and the answering duration is in milliseconds (ms).

Output Data Format

After completing the above data preprocessing, you will get a data.txt file under the dataset namely folder(data directory). Each student sequence contains 6 rows informations as follows:

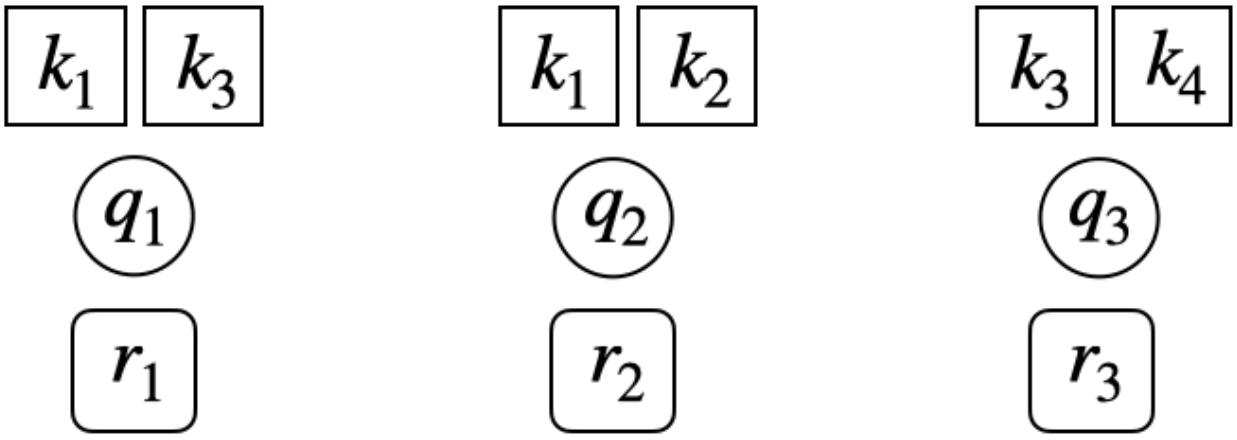
```
User ID, sequence length
Question ID (name)
skill ID (name)
Answer result
Answer submission time
Answering duration
```

Example:

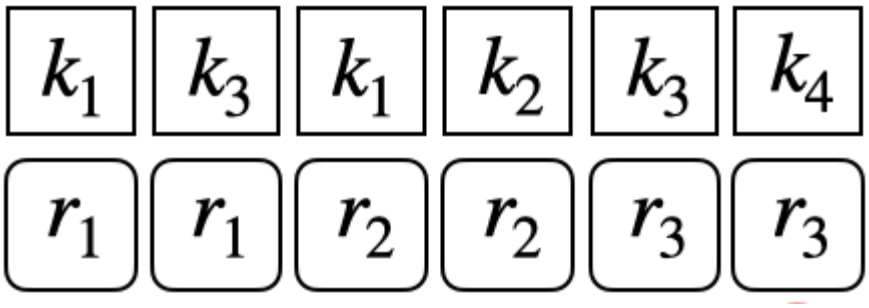
```
50121, 4
106101, 106102, 106103, 106104
7014, 7012, 7014, 7013
0, 1, 1, 1
1647409594000, 1647409601000, 1647409666000, 1647409694000
123, 234, 456, 789
```

Testing Data Format

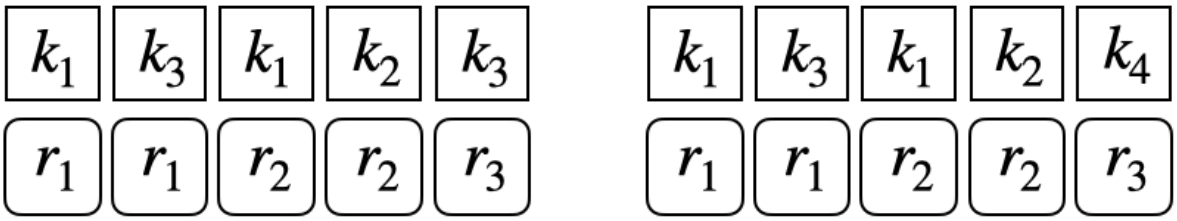
A question may be related to multiple knowledge concepts (KCs). To make the evaluation of pyKT consistent with the real-world prediction scenarios, we train DLKT models on KCs but evaluate them on questions level. To this end, the testing data has different formats in different prediction scenarios. To better understand, we use the below example to introduce various testing data formats. There are three questions ${q_1,q_2,q_3}$, each questions are related to two KCs ${{k_1,k_3},{k_1,k_2},{k_3,k_4}}$.



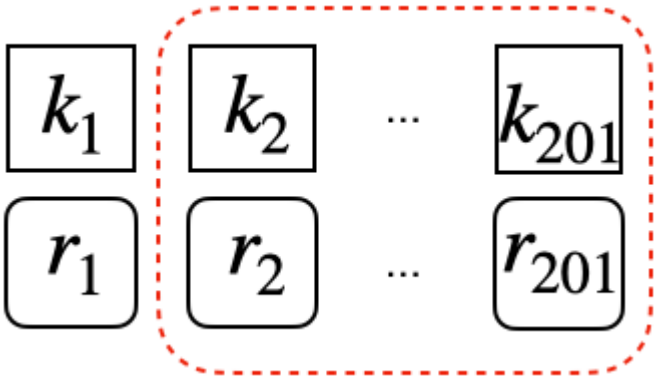
- **Repeat format:** Similar to training data and validation data, the original question-response sequences are expanded into KC level by repeating responses multiple times when a question has more than one KCs, one for each KC.



- **Question level format:** To conduct evaluation on the question level, we split the question-response sequence into KC-response subsequence. For example, for the question q_3 which is corresponding to the ${k3,k4}$. A KC-response subsequence is consist of all the historical interactions and the related KC information as follows:



- **Window format:** If a sequence has more than M interactions, we set a sliding window to keep the sequence length is M. For instance, if $M=200$, the original sequence length is 201, we select the interactions from 1 to 201 as final sequence:



4.3 Add Your Models

4.3.1 Create a New Model File

Our models are all in `pykt/models` directory. When you add a new model, please create a file named `{model_name}.py` in `pykt/models`. You can write your model file using "pykt/models/dkt.py" as a reference.

4.3.2 Init Your Model

You need to add your model in `pykt/models/init_model.py` to init it by changing the `init_model` function.

4.3.3 Add to the Training Process

1. You should change the `model_forward` and `cal_loss` function in `pykt/models/train_model.py` to add your model to the training process, you can refer to other models.
2. Run `wandb_train.py` to train the new model

4.3.4 Add to the Evaluation Process

You can change the `evaluate_model.py` file, change the `evaluate` function to get the repeated knowledge concepts evaluation, change the `evaluate_question` function to get the question evaluation results, change `predict_each_group` and `predict_each_group` to get the multi-step prediction results of accumulative and non-accumulative predictions.

CHAPTER

FIVE

PYKT.MODELS PACKAGE

CHAPTER

SIX

PYKT.DATASETS PACKAGE

CHAPTER

SEVEN

PYKT.PREPROCESS PACKAGE

CHAPTER

EIGHT

PYKT.UTILS PACKAGE

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search