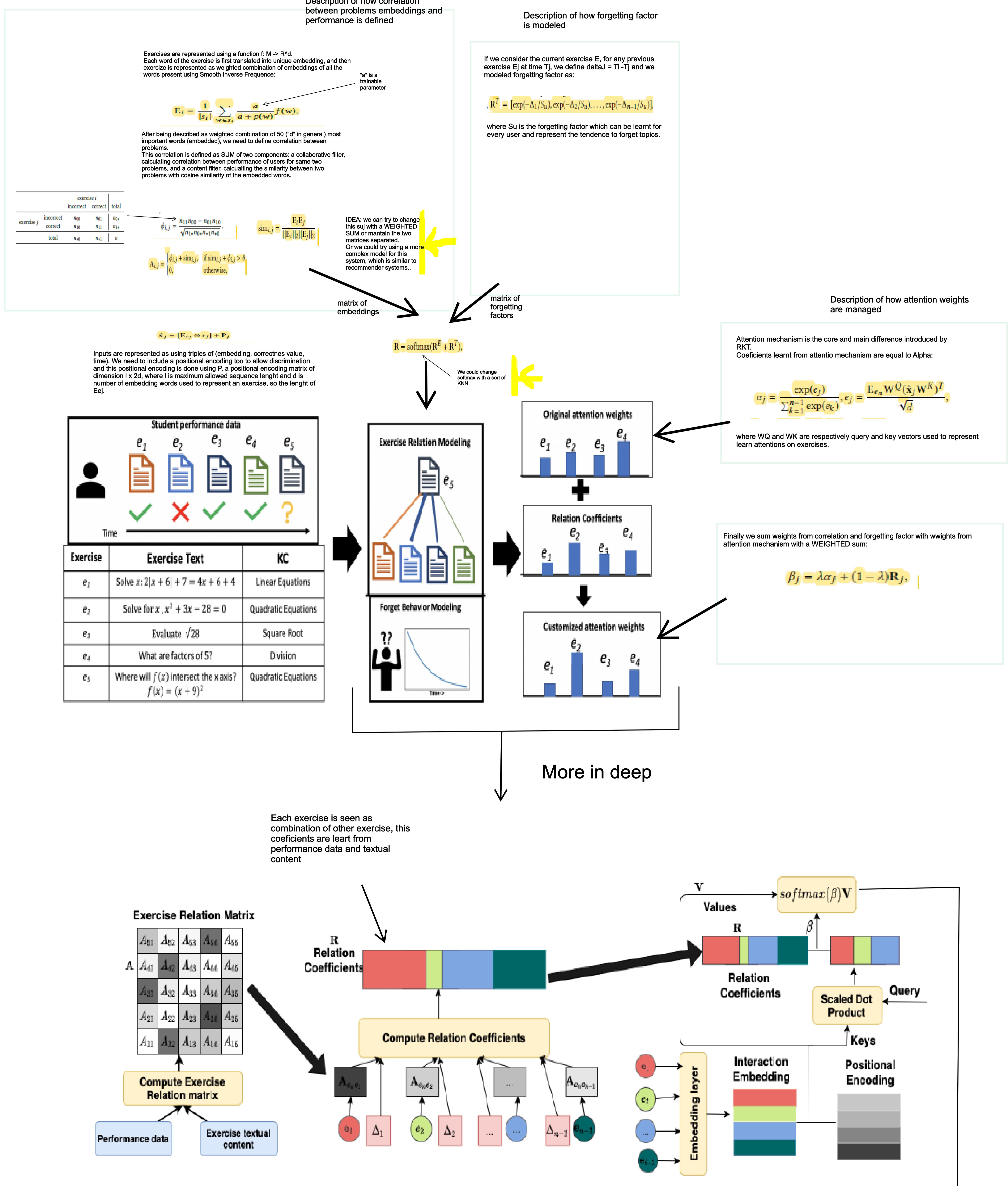


Relation Aware Self-attention Knowledge Tracing



In the end we obtain outputs from weights as:

$$\mathbf{o} = \sum_{j=1}^{n-1} \beta_j \hat{x}_j \mathbf{W}^V,$$

We add Always residual network and apply Normalization and Dropout

And this outputs are then given as input to a Feed Forward Neural Network, to insert possibility of relations between latent dimensions, introducing non-linearities:

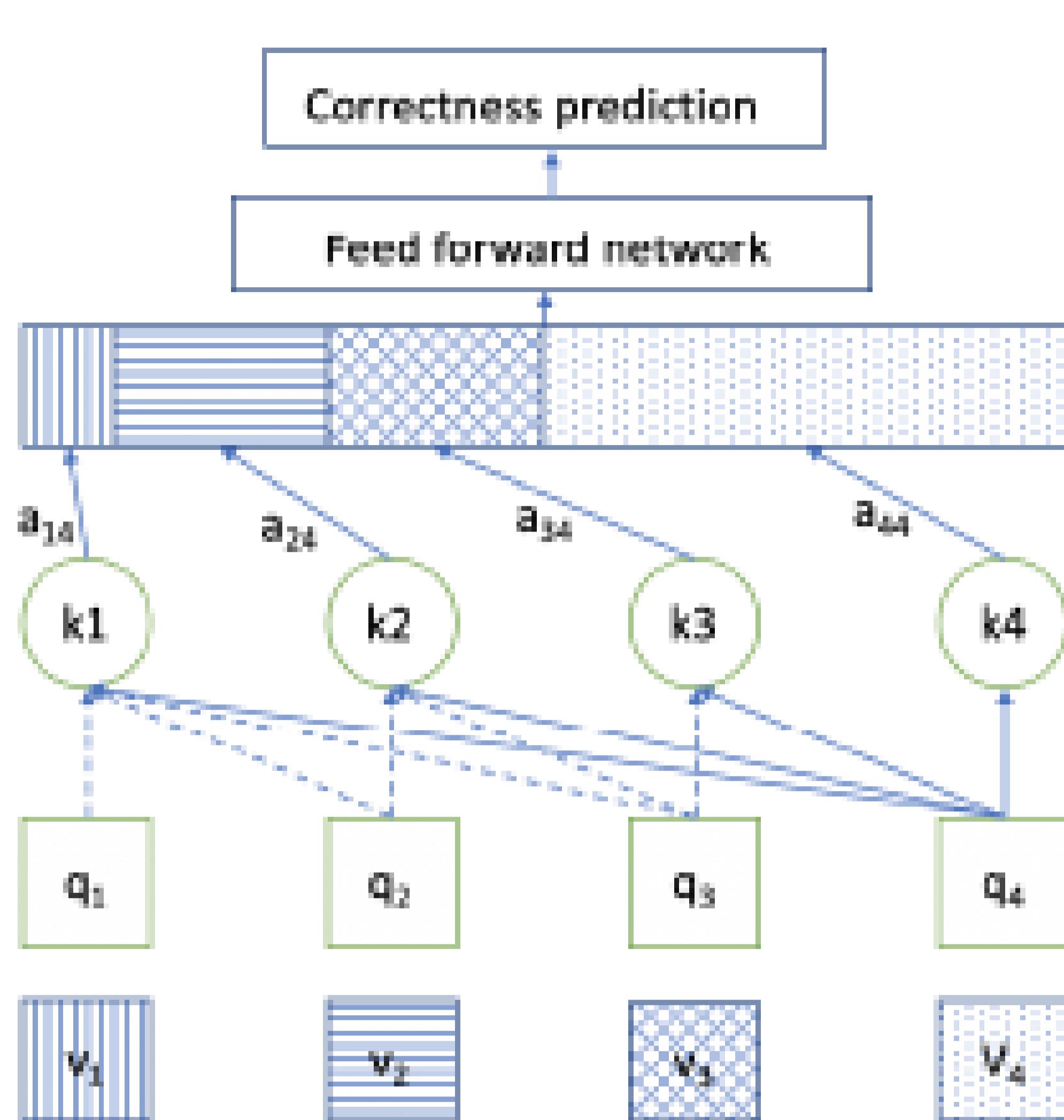
$$\mathbf{F} = \text{ReLU}(\mathbf{o} \mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \mathbf{W}^{(2)} + \mathbf{b}^{(2)},$$

where \mathbf{W} are weight matrices and $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ are Bias vectors.
We add Always residual network and apply Normalization and Dropout

In the end we apply a sigmoid function to a linear layer of \mathbf{F} matrix obtained to generate probability of answering correctly:

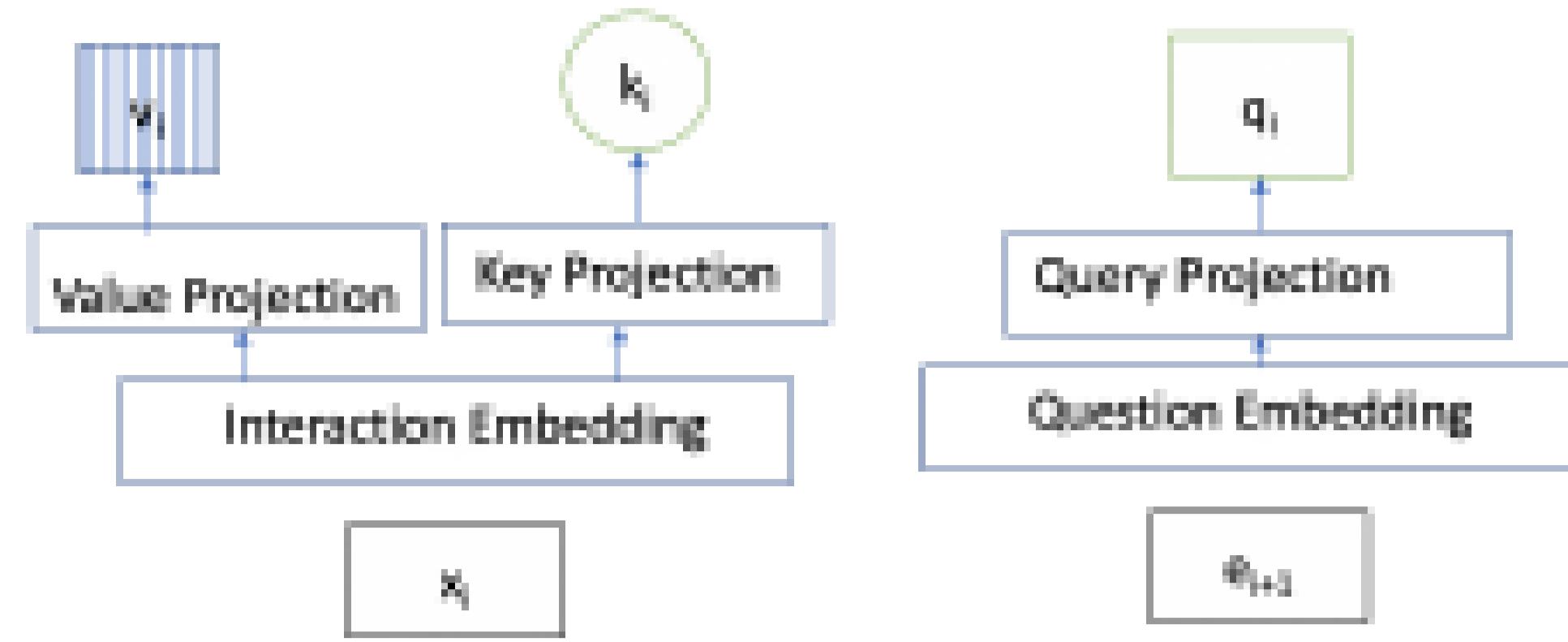
$$p = \sigma(\mathbf{F} \mathbf{W} + \mathbf{b}),$$

SAKT- Self-Attentive model for Knowledge Tracing



In the end we take outputs of Feed Forward network to calculate predicton as:

$$p_i = \text{Sigmoid}(\mathbf{F}_i \mathbf{w} + \mathbf{b}),$$



Input x_i is composed of a couple (exercise, result) and we have a sequence of inputs x_i which constitutes the rel input of the system and the new exercise we want to predict the result x_{i+1} .

The past interactions are embedded using a trained interaction embedding matrix M of dimensions $2E^*d$, to express each exercise as a weighted combination of some terms (embedded concepts). Exercise to predict instead will be embedded with a trained exercise matrix of dimensions E^*d . This will lead to a matrix M and a vector E , we can use them as inputs to a SELF ATTENTIVE layer.

$$\mathbf{Q} = \hat{\mathbf{E}}\mathbf{W}^Q, \mathbf{K} = \hat{\mathbf{M}}\mathbf{W}^K, \mathbf{V} = \hat{\mathbf{M}}\mathbf{W}^V,$$

Where \mathbf{W}_q , \mathbf{W}_k and \mathbf{W}_v are respectively attention matrices for query, key and value and are learnt during training.

In the end we use \mathbf{Q} , \mathbf{K} and \mathbf{V} as input to calculate attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}.$$

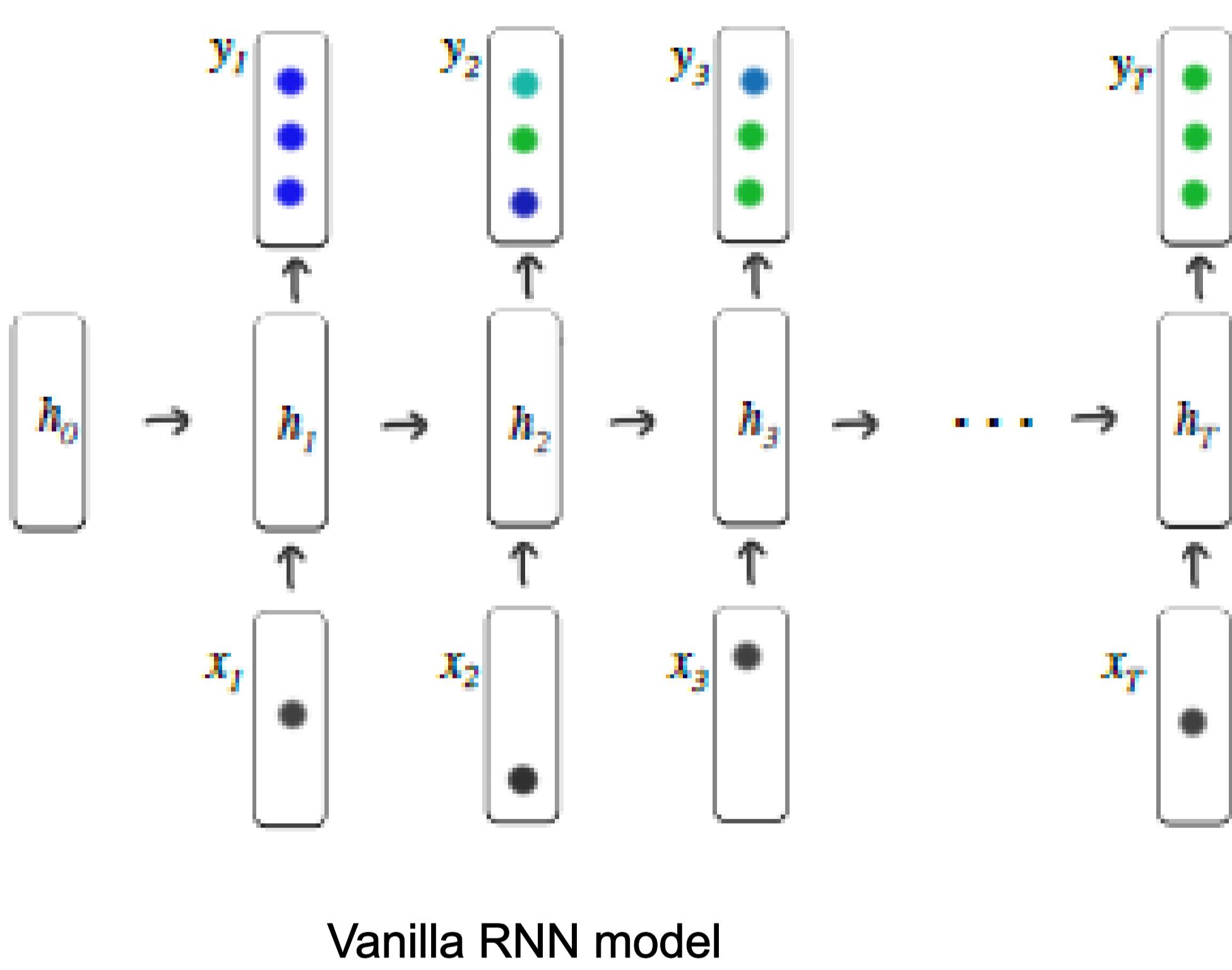
We use Multi Head Attention too, meaning that we calculate different \mathbf{W}_q , \mathbf{W}_k , \mathbf{W}_v to allow detection of different templates.

In the end we take the outputs from different attention heads and use as inputs for a Feed Forward Neural Network to allow non linearities and relations between different heads.

$$\mathbf{F} = \text{FFN}(\mathbf{S}) = \text{ReLU}(\mathbf{S}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)},$$

We use Normalization and add Residual Connections too.

DKT: Deep Learning for Knowledge Tracing



we will apply two different types of RNNs – a vanilla RNN model with sigmoid units and a Long Short Term Memory (LSTM) model.

Inputs must be sequences of fixed lenght input vectors X_t , where we distinguish:

1. with small number M of unique exercises, X_t will be an one-hot encoding of interactions $\{Q_t, A_t\}$. It will have dimension: $2M$, where M is number of unique exercises. So we are describing each exercise as correct or not or not done.
2. with big number M of unique exercises, X_t will be impossible to encode this way, so we will represent each input tuple or with a random vector N_{qa} , with dimension $N << M$ motivated by compressed sensing, which affirms that a K-sparse signal can be recovered from $K \log(d)$ random linear projections.

Outputs instead will be a sequence of values representing probabilities of answering correctly to the corresponding problem.

Optimization objective:

The objective to minimize is the Negative Log Likelihood of observed sequence.equal to:

$$L = \sum_t \ell(\mathbf{y}^T \delta(q_{t+1}), a_{t+1})$$

It is minimized using Stochastic Gradient Descent on minibatches
Dropout is applied to avoid overfitting ONLY for output computation, not for hidden state
Exploding gradient is avoided by truncating the length of gradients with norm over a threshold

We used HIDDEN DIMENSIONALITY of 200, and mini batch-size of 100.

Discovering exercise relationships:

We can use this to discover relationships between exercises (finding knowledge concepts) by assigning a influence J to every directed pair of exercises:

$$J_{ij} = \frac{y(j|i)}{\sum_k y(j|k)},$$

KT-XL: A Knowledge Tracing Model for Predicting Learning Performance Based on Transformer-XL

Prediction Layer

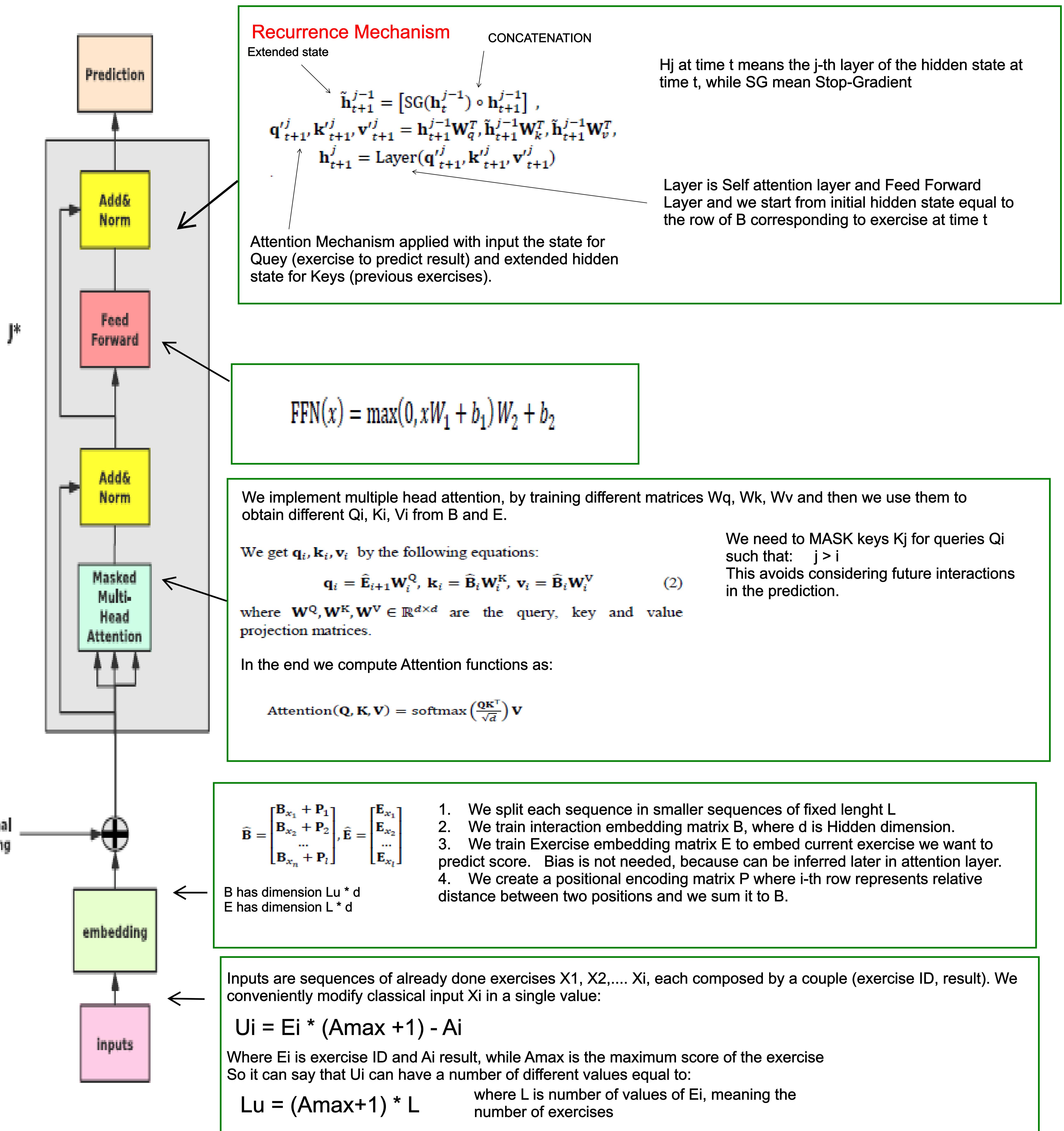
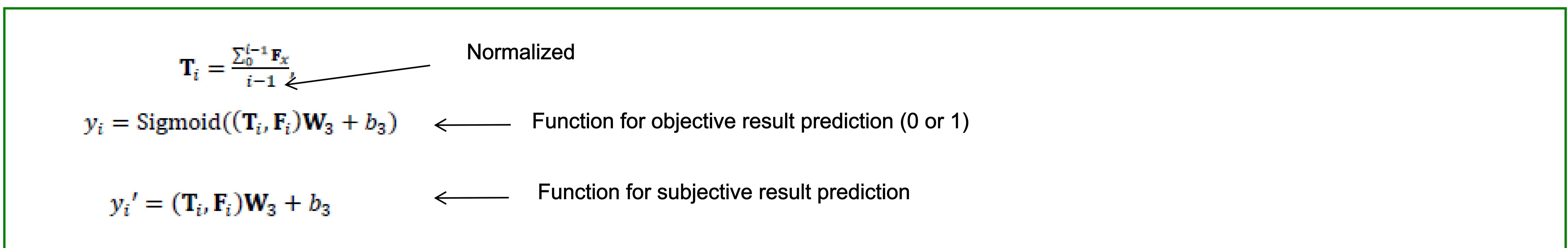


Figure 1: The architecture of KT-XL

DKT_Transformer: Deep Knowledge Tracing with Transformers

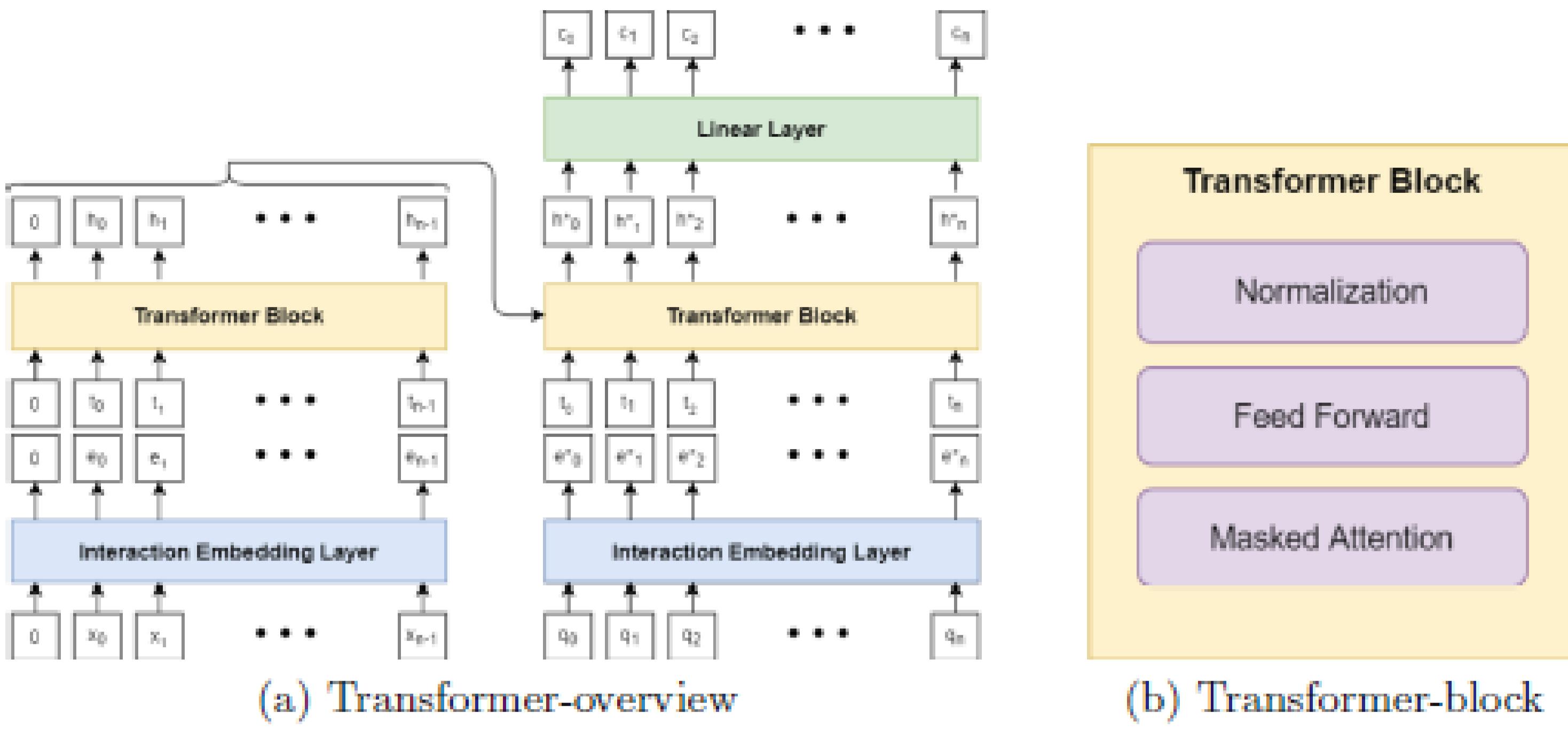


Fig. 1. Transformer architecture

Inputs:

Input are sequence tuples of $X_i = \{ Q_i, C_i \}$ and timestamp T_i , where Q_i represents an univocal exercise, while C_i the correctness of the answer.

GOAL: Predict

$$P(c_{i+1} = 1 | x_0, \dots, x_i, t_0, \dots, t_i, q_{i+1}).$$

Embedding Layers:

X_i will be encoded as an INDEX D_i and passed to an interaction mapping layer:

where:

W is matrix representing each row an exercise and each column the weight of a latent knowledge concept for the exercise

S is vector representation of Skills (Knowledge Concepts)

$$e_i = \text{softmax}(W_{d_i})S$$

weighted sum of all underlying latent skills

Transformers Block - Masked \leftarrow This block is repeated in stack

Attention:

$$q_i = Q e_i, k_i = K e_i, v_i = V e_i$$

Masked Attention Layer extracts query q_i , Key k_i and value v_i from encoding e_i using learnt matrices Q, K, V

$$A_{ij} = \frac{q_i k_j + b(\Delta t_{i-j})}{\sqrt{d_k}}, \forall j \leq i$$

For each previous interaction e_j , from which we extracted key K_j , we calculate attention A_{ij} for question e_i to previous question e_j .

It is computed as sum of two parts:

First part consider the degree of overlapping between latent description of K_j and e_i .
Second part is TIME GAP BIAS

$$h_i = \sum_{j \leq i} \text{softmax}(A_{ij})v_j$$

Hidden representation of exercise q_i is computed as weighted sum of past value representations of e_j multiplied for attention computed before A_{ij} .

Then a FeedForward and a normalization layer are used and residual connections are added.

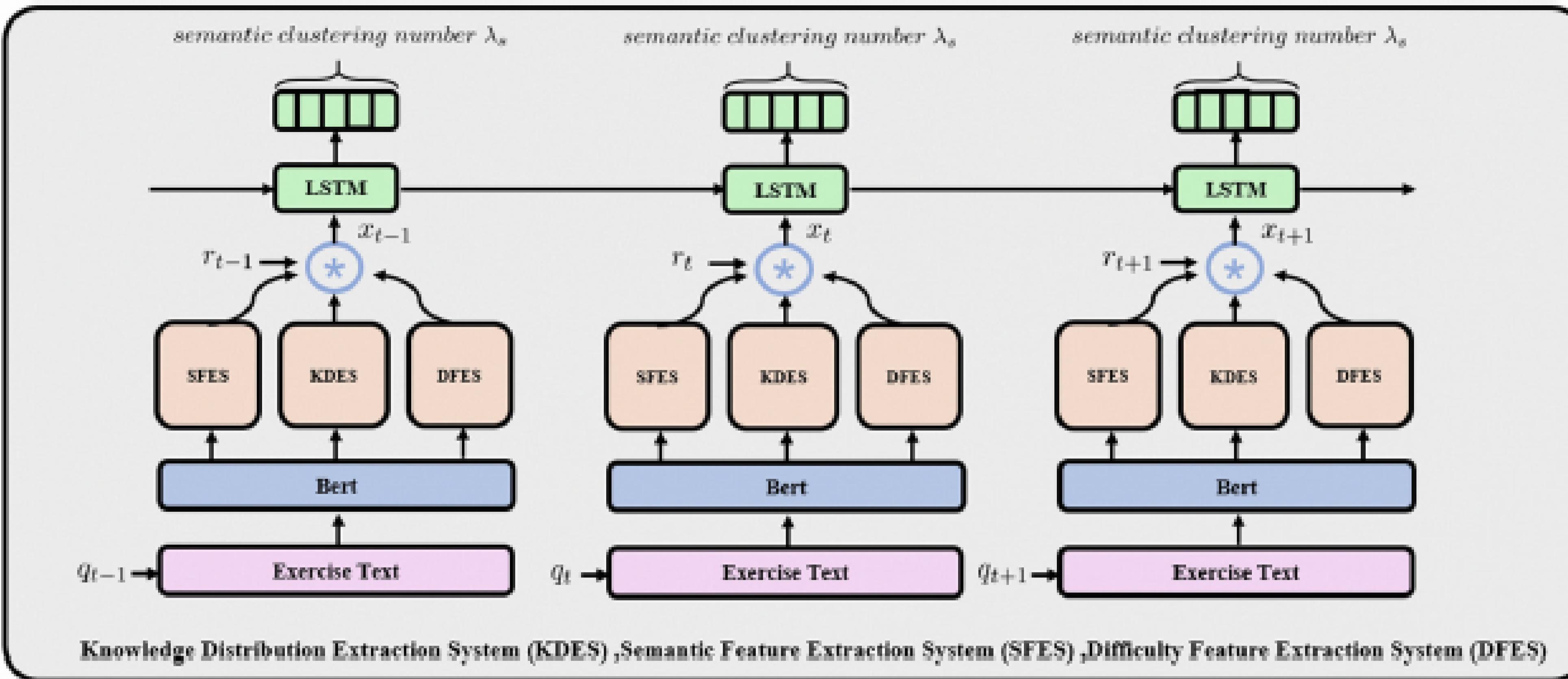
Linear Layer + Loss:

After a stack of transformers block, output is fed to the LINEAR LAYER before calculating this LOSS function:

$$p_{i+1} = \frac{\exp(h_i e_{i+1})}{\exp(h_i e_{i+1}) + \exp(h_i e_{i+1}^*)}$$

$$\text{Loss} = - \sum_i c_{i+1} \log(p_{i+1}) + (1 - c_{i+1}) \log(1 - p_{i+1})$$

Exercise Hierarchical Feature Enhanced Knowledge Tracing



Input encoding:

Exercise recording of a student is a sequence of couples (Q_i, R_i), where Q_i represent the exerciseID and R_i the correctness, and we have to predict future result R_t for exercise Q_t .

The exercise text of Q_t is encoded using BERT into embedding vector V_b .

This vector is then feed to 3 different systems:

One for producing Knowledge Distribution V_t with dimension K . \longrightarrow KDES

One for producing question difficulty D_t . \longrightarrow DFES

One for producing SEMANTIC FEATURES S_t , represented by the one-hot encoding $\text{PHI}(S_t)$. \longrightarrow SFES

The tuple of $(V_t, \text{PHI}(S_t), D_t, R_t)$ will be the input to our sequence model.

Subsystems description:

KDES and DFES are both text classification systems designed to predict respectively Knowledge Distribution and Difficulty of exercise. They are predicted using TextCNN and serve as ground truth.

In KDES we use softmax results, while in DFES we use a neural network to predict difficulty to solve COLD START problem.

Semantic Features Extracion System (SFES) can be considered as an unsupervised clustering problem, solved by using Hierarchical Clustering method calculating COSINE DISTANCE between semantic vectors.

Modeling Process:

We use a RNN (in particular LSTM to preserve long-term dependency) to calculate subsequent hidden state H_t from previous one H_{t-1} and current state X_t equal to tuple $\{V_t, \text{PHI}(S_t), D_t, R_t\}$. $\longrightarrow h_t, c_t = \text{LSTM}(x_t, h_{t-1}, c_{t-1}; \theta_t)$ (1)

TETAt is trainable parameter vector.

Then we use H_t as input to a sigmoid function with trained matrix and bias. $\longrightarrow y_t = \sigma(W_{yh} \cdot h_t + b_y)$ (2)

GOAL: Minimize Negative Log Likelihood of observed sequence of student responses $\longrightarrow \text{loss} = - \sum_t (r_{t+1} * \log(y_i^T \cdot \varphi(s_{t+1})) + (1 - r_{t+1}) * \log(1 - y_i^T \cdot \varphi(s_{t+1})))$ (3)

Results:

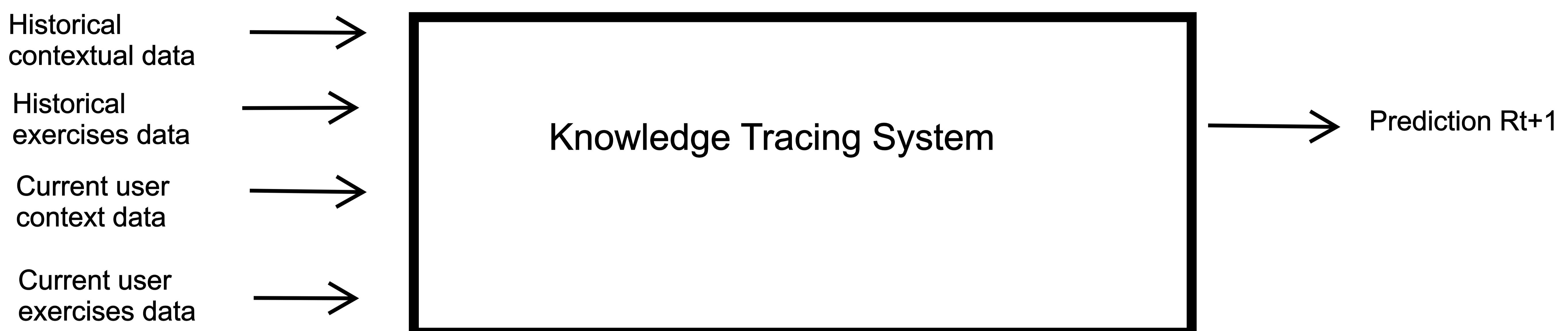
Adding hierarchical features can better represent questions; the reason is that exercises in same cluster have similar semantics, difficulty and knowledge representation

While we have shown the instability of knowledge tracing implicitly due to unpredictability of Difficulty fo exercises.

Personal ideas/suggestions:

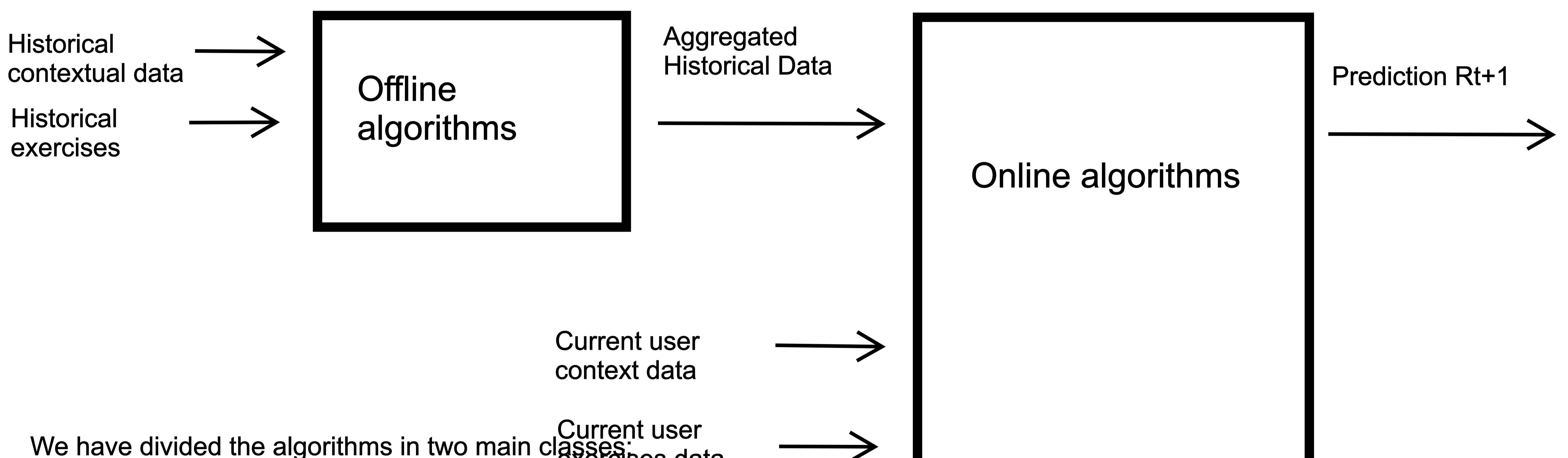
Modular Generalization of Knowledge Tracing:

We can try define a general model to represent the common characteristics a Knowledge Tracing system should have in order to provide a structure for the models allowing to reduce the correctness prediction problem to a composition of "smaller" problems, according to design pattern "Divide Et Impera":



A generic model for KT has different inputs, which can be divided in three subsets:

- 1) Historical exercises data, which in general represent the total amount of available data about the interaction of users and exercises in the past and their characteristics. This is a dataset to be maintained offline.
- 2) Historical contextual data, which represent the total amount of available data about the context of past data. It can contain aggregated extracted information, computed offline. This is a dataset to be maintained offline.
- 3) Current user exercise data, they represent previous interactions between user and exercises. They need at least to contain correctness value and possibly timestamp, but they can contain many other useful data.
- 4) Current user contextual data, which contain any data known about the user, such as age, field of study, and hypothetically should contain Knowledge state estimation.



We have divided the algorithms in two main classes:

- 1) Offline algorithms, which take as input Historical data (both generic and user specific), and return in general aggregated data which will be used by online algorithms.

These algorithms can be performed only periodically and not each time an user answer to an exercise, due to the memory and time performances.

They contain aggregated information learnt from past and should try to detect patterns and elements to help online algorithms.

- 2) Online algorithms, which should be evaluated each time the Current user answer a new exercise; they are personalized algorithms, which use as input both personalisez Current user data and gggregated data from offline algorithms.

In general they should be less computational and memory expensive, because they should be evaluated fast to allow online decisions for the application.

Offline algorithms:

From literature we have many examples of offline algorithms already efficiently developed:

- 1) Exercise Relation Modeling from RKT is a clear example of offline algorithms in which we use as input both Historical Exercise Data (called in the paper performance data, and which are simply the tuples {Exercise, Correctness, Timestamp}) and Historical Context Data (in particular they use exercise textual content).
The outputs are the coefficients A_{ij} representing correlation between exercises i and j.
Actually this method can be seen as an hybrid combination (linear combination of coefficients) of 2 different methods: a collaborative filter using the Phi coefficient between exercise performances, and a content filter using cosine similarity to calculate content similarity.
In the end these two methods are simply summed but in reality they could produce two different Exercise Relation Matrices.

2)