# Behavior-Aware Item Modeling via Dynamic Procedural Solution Representations for Knowledge Tracing

**Jun Seo**[1]* **Sangwon Ryu**[1]* **Heejin Do**[3]† **Hyounghun Kim**[1,2] **Gary Geunbae Lee**[1,2]†
[1]GSAI, POSTECH [2]CSE, POSTECH
[3]ETH Zurich, ETH AI Center
{sjin4861, ryusangwon, h.kim, gblee}@postech.ac.kr
heejin.do@ai.ethz.ch

## Abstract

Knowledge Tracing (KT) aims to predict learners' future performance from past interactions. While recent KT approaches have improved via learning item representations aligned with Knowledge Components, they overlook the procedural dynamics of problem solving. We propose Behavior-Aware Item Modeling (BAIM), a framework that enriches item representations by integrating dynamic procedural solution information. BAIM leverages a reasoning language model to decompose each item's solution into four problem-solving stages (i.e., understand, plan, carry out, and look back), pedagogically grounded in Polya's framework. Specifically, it derives stage-level representations from per-stage embedding trajectories, capturing latent signals beyond surface features. To reflect learner heterogeneity, BAIM adaptively routes these stage-wise representations, introducing a context-conditioned mechanism within a KT backbone, allowing different procedural stages to be emphasized for different learners. Experiments on XES3G5M and NIPS34 show that BAIM consistently outperforms strong pretraining-based baselines, achieving particularly large gains under repeated learner interactions. Code and data are available in: sjin4861/BAIM.

## 1 Introduction

Knowledge Tracing (KT) aims to predict a learner's future performance (i.e., whether they can correctly solve a new problem) from historical interaction data (Corbett and Anderson, 1994). A critical factor in KT is the quality of item representations, as they affect a model's ability to capture dependencies among items and to update learners' knowledge states from observed responses. While recent deep learning-based KT models primarily focus on improving temporal prediction through sequence modeling (Huang et al., 2023; Xu et al., 2023; Huang et al., 2024), the representation of individual items remains largely underexplored. As a result, item identifiers are often mapped to randomly initialized embeddings learned solely from sparse and highly imbalanced interaction data, making it difficult to acquire robust semantic item representations (Krivich et al., 2025).

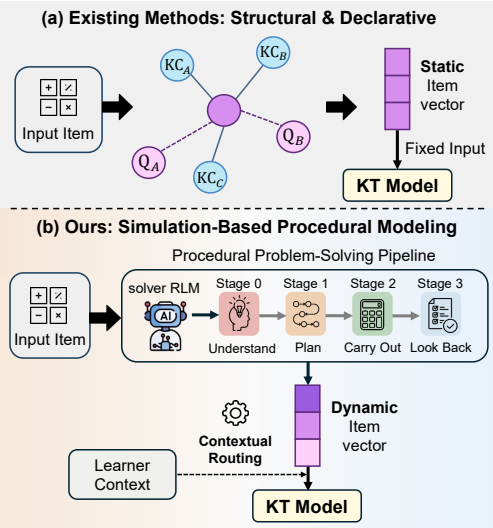



Figure 1: Comparison between conventional item modeling and BAIM. (a) Prior KT models rely on static item embeddings derived from item–KC structures. (b) BAIM instead generates procedural, context-aware item representations via Polya-based reasoning simulation.

To address this limitation, recent work has proposed pre-trained item embedding methods that encode structural relationships between items and associated Knowledge Components (KCs) (Liu et al., 2020; Wang et al., 2022; Song et al., 2022; Wang et al., 2024a; Ozyurt et al., 2024). However, these approaches primarily encode declarative components into static representations, overlooking the procedural dynamics of the problem-solving process. In practice, solving a problem involves multiple stages—such as interpreting the problem,

*Equal contribution.
†Corresponding authors.



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 22428–22443
July 2-7, 2026 ©2026 Association for Computational Linguistics

setting solution strategies, and executing calculations—each reflecting distinct procedural demands, even for items associated with the same underlying concept (Schoenfeld, 2014). Moreover, the relative importance of these stages varies with a learner's knowledge state and interaction history (Schoenfeld and Herrmann, 1982). Therefore, capturing such learner-dependent variability requires item representations that move beyond static embeddings toward adaptive modeling of procedural solution processes.

In this paper, we propose **Behavior-Aware Item Modeling (BAIM)**, a novel framework that represents items through their problem-solving processes and adapts these representations to individual learners (Figure 1). Grounded in Polya's four-stage problem-solving process (Pólya, 1957) (i.e., Understanding, Planning, Carrying Out, and Looking Back), BAIM decomposes each item into structured solution stages. For each stage, BAIM leverages a reasoning language model (RLM) to derive stage-wise solution representations that capture rich latent embedding trajectories beyond surface-level item content. To adaptively leverage these stage-wise representations, BAIM introduces a context-conditioned routing mechanism that emphasizes the most informative problem-solving stage based on the learner's prior interactions. Notably, BAIM avoids auxiliary network pre-training by using one-time RLM inference and internalizes the adaptive routing mechanism directly into the KT model for unified end-to-end training.

We evaluate BAIM on the XES3G5M (Liu et al., 2023b) and NIPS34 (Wang et al., 2020) benchmarks, where it consistently outperforms strong pretraining-based item embedding methods. In particular, BAIM exhibits clear advantages in repeated problem-solving attempts, highlighting its ability to adapt item representations to evolving learner–item interactions. Further analysis shows that leveraging the embedding trajectory yields richer and more transferable representations than using final-layer or text-only encodings. In addition, BAIM achieves faster performance gains in low-data regimes, highlighting its effectiveness under realistic educational constraints. Our main contributions are summarized as follows:

- We propose BAIM, a stage-based item modeling framework grounded in Polya's problem-solving theory, representing items through structured problem-solving stages.
- We derive stage-level representations from embedding trajectories of an RLM, capturing cognitive signals beyond surface semantics.
- We introduce a context-conditioned routing mechanism to adaptively integrate stage-level solution representations according to the learner's interaction history.
- Extensive experiments demonstrate BAIM's robustness and adaptability in realistic settings, including repeated problem-solving attempts and low-data regimes.

## 2 Related Work

**Item Representation learning in KT** Recent work on item representation learning in KT has focused on enriching item representations by leveraging KCs and their relational dependencies. PEBG (Liu et al., 2020) introduced bipartite graph–based representations that explicitly encode item–KC interactions. Subsequent self-supervised approaches extended this direction by leveraging KC-anchored relational structures through diverse learning objectives, including contrastive and relation-based pre-training (Wang et al., 2022; Song et al., 2022; Wang et al., 2024a; Lee et al., 2024). More recent work explores generative approaches; KCQRL (Ozyurt et al., 2024) leverages LLM-generated step-by-step solutions to automatically annotate KCs and learn enriched item representations via contrastive objectives. Despite their effectiveness, they embed items as *static vectors* that primarily reflect declarative knowledge or structural similarity, leaving the procedural dynamics of problem-solving largely unmodeled. Moreover, they typically require additional network pre-training of item representations when new items are introduced. To address these limitations, we model item representations via procedural solution processes, capturing problem-solving dynamics beyond KC-centric structures, enabling adaptive and context-aware representations without additional pre-training as data evolves.

**Deep Knowledge Tracing** KT research has focused on developing neural architectures for modeling learners' interaction sequences. DKT (Piech et al., 2015) introduced LSTM-based sequence modeling, while qDKT (Sonkar et al., 2020) highlighted the necessity of item-level distinctions among problems sharing the same KC. With the adoption of Transformer (Vaswani et al., 2017)

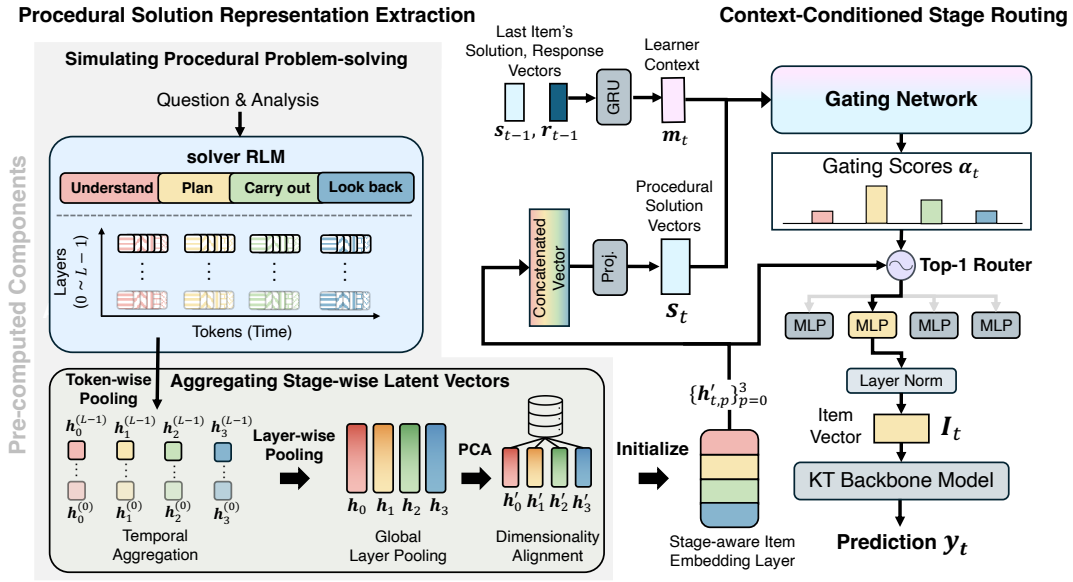



Figure 2: Overview of the proposed BAIM framework. **Left:** A solver RLM simulates Polya-based procedural problem solving for each item and extracts stage-wise latent representations. **Center:** Stage-aware procedural embeddings yield procedural solution vectors, and a learner context encoder encodes the learner's interaction history; both serve as inputs to the gating network. **Right:** A context-conditioned stage routing module adaptively selects the most relevant problem-solving stage to produce a learner-conditioned item representation for KT.

architectures, AKT (Ghosh et al., 2020) further advanced the field by combining monotonic decay attention for sequence modeling with Rasch-based embeddings for enhanced item representation. Subsequent work has emphasized simplicity and robustness in model design; for example, simpleKT (Liu et al., 2023a) and sparseKT (Huang et al., 2023) achieved competitive performance by emphasizing architectural simplicity and sparse attention mechanisms, respectively. In parallel, efforts to improve interpretability have led to cognitively grounded designs such as QIKT (Chen et al., 2023), which adopts item-centric cognitive representations leveraging associated KCs as auxiliary information, and incorporates an item response theory-based prediction layer. To evaluate our item modeling approach, we integrate BAIM into the item representation components of these KT backbones, while preserving their original sequence modeling and prediction mechanisms.

## 3 Problem Statement

Following prior work (Sonkar et al., 2020), we adopt the standard *item-level* formulation of KT, where the objective is to predict a learner's response to a given item at time step $t$ based on the learner's historical interaction sequence. A learner's interaction history is represented as a temporal sequence $X = \{x_0, x_1, \ldots, x_{t-1}\}$, where the $j$-th interaction $x_j$ is defined as a 2-tuple $x_j = (I_j, r_j)$. Here, $I_j$ denotes a unique item identifier, and $r_j \in \{0, 1\}$ indicates whether the learner answered the question of the item correctly. Given the historical interactions $\{x_j\}_{j=0}^{t-1}$ and the current item $I_t$, a KT model estimates the probability that the learner answers the item correctly:

$$y_t = P(r_t = 1 \mid I_t, \{x_j\}_{j=0}^{t-1}). \tag{1}$$

The model is trained by minimizing the binary cross-entropy loss between the predicted probability $y_t$ and the observed response $r_t$:

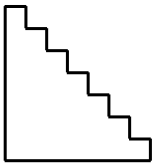
$$\mathcal{L}_{\text{KT}} = -\sum_t \left[ r_t \log y_t + (1 - r_t) \log(1 - y_t) \right]. \tag{2}$$

In this item-centric formulation, the item identifier $I_j$ serves as the primary modeling unit, and all predictions and losses are defined with respect to item responses. KCs, when available, are treated as contextual information rather than independent prediction targets.

## 4 Behavior-Aware Item Modeling (BAIM)

Existing item representation learning approaches primarily rely on KC tags, which, from the ACT-R perspective, represent *declarative knowledge* (Anderson, 1996). However, this focus overlooks *procedural knowledge*—the process and capability in-

volved in solving problems. To address this limitation, we propose BAIM, which captures both aspects by explicitly modeling the act of solving itself. BAIM operates by integrating procedural solution representations with item- and learner-conditioned contextual signals. First, it extracts procedural solution representations for each item by decomposing the solution process into structured problem-solving stages following Polya's framework, including *Understand*, *Plan*, *Carry Out*, and *Look Back*. Second, BAIM introduces a *Context-Conditioned Stage Routing* mechanism that adaptively determines which problem-solving stage should be emphasized for a given item, conditioned on the learner's interaction context. Through this routing mechanism, BAIM aligns item-level procedural characteristics with the learner's context, enabling more fine-grained personalized diagnosis and prediction. The overall process is illustrated in Figure 2.



**Question**. The side view of a toy staircase is shown above. Each step is 10 cm wide and 8 cm high. What is the perimeter of the staircase side?

**Analysis.** Using a geometric translation strategy, the total horizontal length is 7 × 10 and the total vertical length is 7 × 8. The perimeter is 2 × (70 + 56) = 252 cm.

| ($p$) Stage | Stage-wise Solution Description |
|---|---|
| **(0) Understand** | Each step is 10 cm wide (tread) and 8 cm high (riser). There are 7 steps. The goal is to calculate the perimeter of the staircase side view. |
| **(1) Plan** | Use the translation method: total horizontal segments $= 7 \times 10$, total vertical segments $= 7 \times 8$. The perimeter is $2\times$ (total horizontal + total vertical). |
| **(2) Carry Out** | $7 \times 10 = 70$; $7 \times 8 = 56$; $70 + 56 = 126$; $126 \times 2 = 252$. |
| **(3) Look Back** | The perimeter is 252 cm, which matches the analysis and accounts for all outer edges of the staircase. |

Figure 3: Example of a staircase-shaped geometry problem from the XES3G5M dataset and its stage-wise solution description.

### 4.1 Procedural Solution Representation Extraction

**Simulating Procedural Problem-solving.** The first step of BAIM simulates the problem-solving process for the given item. We employ an RLM as a solver that generates a structured problem-solving process according to Polya's four-stage framework, as illustrated in Figure 3. Given an item and its accompanying analysis, the solver RLM generates a problem-solving process together with a structured output that delineates four problem-solving stages after internal reasoning. Each stage $p \in \{0, 1, 2, 3\}$ corresponds to a contiguous token span $(T_{s,p}, T_{e,p})$. During this process, the RLM naturally generates token- and layer-wise latent vectors, which form the basis for subsequent stage-wise aggregation. Details of the prompting strategy are provided in Appendix H.

**Aggregating Stage-wise Latent Vectors.** We construct procedural representations for KT by aggregating embedding trajectories produced by an RLM at different granularities. Specifically, we first perform temporal aggregation by aligning RLM's token-level hidden states along the generation timeline with predefined problem-solving stages, motivated by prior work that demonstrates the effectiveness of aggregating internal hidden states over the generated sequence for tasks such as out-of-distribution detection and self-evaluation (Wang et al., 2024b, 2025b). We then apply global pooling over the resulting embedding trajectories to integrate procedural signals captured across the full trajectory.

**(a) Temporal Aggregation.** For each problem-solving stage $p$ and each transformer layer $l$, we aggregate token-level hidden states within the stage-specific token span $[T_{s,p}, T_{e,p}]$ using mean pooling. This operation summarizes the latent state of the solver corresponding to stage $p$ at depth $l$:

$$\mathbf{h}_p^{(l)} = \frac{1}{T_{e,p} - T_{s,p} + 1} \sum_{k=T_{s,p}}^{T_{e,p}} \mathbf{h}_k^{(l)}, \quad \mathbf{h}_k^{(l)} \in \mathbb{R}^{D_{\text{solver}}}. \quad (3)$$

where $\mathbf{h}_p^{(l)}$ denotes a latent vector associated with stage $p$ at layer $l$ and is treated as an intermediate quantity rather than a final representation.

**(b) Global Layer Pooling.** The stage-wise latent vectors obtained above still retain layer-specific variations. Latent vectors extracted from different layers capture complementary aspects of the solution process, reflecting how information is progressively transformed across the model depth. Recent studies suggest that relying on a single or shallow subset of layers may overlook useful procedural information encoded in intermediate representations (Tang and Yang, 2024; Skean et al., 2025). Accordingly, we perform entire layer pooling by

averaging across all $L$ transformer layers:

$$\mathbf{h}_p = \frac{1}{L} \sum_{l=0}^{L-1} \mathbf{h}_p^{(l)}. \tag{4}$$

This operation yields a unified stage-level latent summary that integrates information distributed throughout the entire model.

**(c) Dimensionality Alignment.** Since the dimensionality of $\mathbf{h}_p$ exceeds that of standard KT backbones, we apply PCA (Bishop and Nasrabadi, 2006) to reduce it to 768 dimensions for compatibility with prior work (Ozyurt et al., 2024). The resulting vectors are used as the stage-level representations for item $I_t$, denoted as $\{\mathbf{h}'_p\}_{p=0}^{3}$. These representations are precomputed for all items and used to initialize the item embedding layer of the KT model, providing each item with stage-aware procedural priors. After initialization, the embedding corresponding to item $I_t$ and stage $p$, denoted as $\mathbf{h}'_{t,p}$, is optimized jointly with the training objective rather than being frozen.

### 4.2 Context-Conditioned Stage Routing

Not all problem-solving stages are equally informative in practice. Prior work in mathematical problem solving and cognitive load theory suggests that the cognitive demands and diagnostic relevance of each stage vary across problems and depend on the items' and learners' context (Sweller, 1988; Schoenfeld, 2014). Therefore, we propose a *Context-Conditioned Stage Routing* mechanism that adaptively adjusts the emphasis placed on different problem-solving stages based on the learner context. All architectural specifications, including dimensionalities and parameter configurations, are provided in Appendix A.

**Procedural Solution Encoding.** Given the stage-level representations $\{\mathbf{h}'_{t,p}\}_{p=0}^{3}$ for item $I_t$, we aggregate them to form a single *procedural solution representation*:

$$\mathbf{s}_t = f_{\text{proj}} \left( \mathbf{h}'_{t,0} \oplus \mathbf{h}'_{t,1} \oplus \mathbf{h}'_{t,2} \oplus \mathbf{h}'_{t,3} \right), \tag{5}$$

where $\oplus$ denotes concatenation and $f_{\text{proj}}(\cdot)$ denotes a learnable projection function. The resulting vector $\mathbf{s}_t$ provides a unified procedural encoding of item $I_t$ and serves as the item-side input to the context-conditioned stage routing mechanism.

**Learner Context Encoding.** To condition stage routing on the learner's historical interaction patterns, we encode the learner's context into a latent context vector $\mathbf{m}_t$. Given the learner's interactions history $\{(I_j, r_j)\}_{j=0}^{t-1}$, the learner context is updated using a Gated Recurrent Unit (GRU) (Chung et al., 2014):

$$\mathbf{m}_t = \text{GRU}\left(\mathbf{m}_{t-1}, \mathbf{W}_{\text{in}} \left[\mathbf{s}_{t-1} \oplus \mathbf{r}_{t-1}\right]\right), \tag{6}$$

This formulation provides a contextual signal that conditions the subsequent stage routing process.

**Top-1 Stage Routing Mechanism.** Given the procedural solution vector $\mathbf{s}_t$ and the learner context vector $\mathbf{m}_t$, the routing module computes gating scores $\boldsymbol{\alpha}_t \in \mathbb{R}^4$ by jointly conditioning on both signals:

$$\boldsymbol{\alpha}_t = \mathbf{W}_{\text{gate}} \left[\mathbf{s}_t \oplus \mathbf{m}_t\right] + \mathbf{b}_{\text{gate}}. \tag{7}$$

Conditioned on the joint item–learner context, we adopt a Top-1 routing strategy, selecting the stage with the highest gating score (i.e., $k^* = \arg\max \boldsymbol{\alpha}_t$), thereby assigning different importance to problem-solving stages.

**Stage-Specific Expert Transformation.** The selected stage-level representation $\mathbf{h}'_{t,k^*}$ is transformed by a stage-specific expert to produce the final context-conditioned item representation:

$$\mathbf{I}_t = \text{LayerNorm}\left(\text{MLP}_{k^*}\left(\mathbf{h}'_{t,k^*}\right)\right). \tag{8}$$

The resulting vector $\mathbf{I}_t$ constitutes the final context-conditioned item representation and is consumed by the downstream KT backbone.

### 4.3 Objective Function

BAIM is trained under the standard item-level KT objective defined in Section 3. To prevent routing collapse, we additionally apply a load-balancing regularization term following Switch Transformers (Fedus et al., 2022). Specifically, given the gating scores $\boldsymbol{\alpha}_t$, we compute gating probabilities $\mathbf{p}_t = \text{Softmax}(\boldsymbol{\alpha}_t)$. Let $\bar{p}_j$ denote the batch-averaged probability of selecting stage $j$, load-balancing loss is defined as:

$$\mathcal{L}_{\text{LB}} = \sum_{j=0}^{3} \left( \bar{p}_j - \frac{1}{4} \right)^2. \tag{9}$$

We define the final training objective as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{KT}} + \lambda \mathcal{L}_{\text{LB}}. \tag{10}$$

| Attribute | Datasets | |
|---|---|---|
| | XES3G5M | NIPS34 |
| # Students | 18.07K | 4.92K |
| # Questions | 7.65K | 948 |
| # KCs | 865 | 57 |
| # Interactions | 5.55M | 1.38M |
| Question Meta | Text + Image (CN) | Image-only (EN) |
| Analysis Meta | Provided | Not provided |

Table 1: Dataset statistics and metadata availability. Question Meta indicates the original format of question content, and Analysis Meta denotes whether analytical explanations are provided.

## 5 Experimental Setup

**Datasets** We evaluate BAIM on two real-world mathematical KT benchmarks that provide publicly available item-level metadata: **XES3G5M** (Liu et al., 2023b)[1] and **NIPS34** (Wang et al., 2020).[2] XES3G5M serves as our primary large-scale benchmark, and we additionally evaluate on NIPS34 to validate its robustness. Dataset statistics and metadata characteristics are summarized in Table 1. Due to differences in item and analysis metadata formats, we apply dataset-specific preprocessing to enable reasoning-based item modeling. Further details are provided in Appendix D.

**Solver RLM** We use Qwen3-VL-32B-Thinking (Bai et al., 2025) as the solver RLM to generate procedural solution representations for items. The solver is used offline to extract stage-level representation for each item. The model comprises $L = 65$ transformer layers with a hidden dimensionality of $D_{\text{solver}} = 5{,}120$.

**Item Representation Baselines** We compare BAIM against two well-established item representation baselines while keeping the underlying KT backbone architectures fixed. The **Default** setting uses randomly initialized item- and KC-level embeddings that are trained end-to-end together with each backbone. For **pre-trained baselines**, we reproduce 768-dimensional item embeddings using the official implementations of `PEBG`[3] and `KCQRL`.[4] Implementation details and reproduction-specific adaptations are provided in Appendix B.

**Backbone KT Models** We select five representative KT backbones—**AKT**, **qDKT**, **QIKT**, **simpleKT**, and **sparseKT**—which achieved strong performance in the KCQRL benchmark, and adapt them to our framework for evaluation. For each backbone, we replace the *Item Representation Module* with BAIM while keeping all other architectural components unchanged. Backbone-specific integration details are in Appendix C.

**Evaluation** We evaluate the effectiveness of item representations on each dataset by measuring their impact on KT performance using AUC. Results for which the mean or standard deviation is reported are obtained via 5-fold cross-validation, with the random seed fixed to 42 for reproducibility.

**Training Details** All models are trained with a batch size of 128, an embedding size of 256, and a dropout rate of 0.1. We fix the learning rate to $1 \times 10^{-4}$ and adopt default hyperparameter settings from the `pyKT` library.[5] We set the loss weighting coefficient $\lambda$ to 0.01, following the setting used in Switch Transformers (Fedus et al., 2022).

## 6 Results

**Main Results** To assess the effectiveness of BAIM, we compare BAIM against strong item representation baselines across five representative KT backbones on XES3G5M and NIPS34, demonstrating clear improvements in KT performance. As shown in Table 2, BAIM achieves the highest AUC across all KT architectures on the XES3G5M dataset. Compared to both randomly initialized embeddings (`Default`) and pre-trained item representations (`PEBG` and `KCQRL`), BAIM yields performance gains across diverse backbone designs. In particular, BAIM achieves the largest gain on `sparseKT`, improving AUC from 80.37 to 83.21, suggesting that behavior-aware item representations are especially effective when combined with sparse attention mechanisms. Notably, the advantages of BAIM extend beyond large-scale, text-rich settings. On the smaller-scale NIPS34 dataset, which differs substantially in both scale and metadata format, BAIM continues to outperform both the `Default` and `PEBG` baselines across all KT backbones. Overall, these results indicate that BAIM generalizes well across datasets with heterogeneous characteristics, highlighting the robustness of behavior-aware item modeling beyond specific data conditions.

**Stage-Level Dynamics under Repeated Interactions** To evaluate the adaptability of item

[1] https://github.com/ai4ed/XES3G5M.git
[2] https://www.eedi.com/research
[3] https://github.com/ApexEDM/PEBG.git
[4] https://github.com/oezyurty/KCQRL.git
[5] https://pykt.org

| Embed \ Model | Knowledge Tracing Backbone Architecture | | | | |
|---|---|---|---|---|---|
| | AKT | qDKT | QIKT | simpleKT | sparseKT |
| **XES3G5M** | | | | | |
| Default | 81.56 ± 0.06 | 81.69 ± 0.04 | 81.67 ± 0.01 | 81.26 ± 0.01 | 80.37 ± 0.04 |
| PEBG | 82.79 ± 0.04 (+1.23) | 82.16 ± 0.04 (+0.47) | 82.00 ± 0.02 (+0.33) | 82.51 ± 0.02 (+1.25) | 82.63 ± 0.07 (+2.26) |
| KCQRL | 82.67 ± 0.03 (+1.11) | 81.94 ± 0.03 (+0.25) | 81.85 ± 0.02 (+0.18) | 82.48 ± 0.02 (+1.22) | 82.61 ± 0.11 (+2.24) |
| **BAIM** (Ours) | **83.00** ± 0.04 (**+1.44**) | **82.43** ± 0.02 (**+0.74**) | **82.17** ± 0.05 (**+0.50**) | **82.84** ± 0.01 (**+1.58**) | **83.21** ± 0.10 (**+2.84**) |
| **NIPS34** | | | | | |
| Default | 79.89 ± 0.07 | 79.24 ± 0.08 | 79.95 ± 0.07 | 79.90 ± 0.01 | 79.30 ± 0.08 |
| PEBG | 80.10 ± 0.02 (+0.21) | 80.10 ± 0.03 (+0.86) | 80.15 ± 0.03 (+0.20) | 79.96 ± 0.01 (+0.06) | 80.21 ± 0.17 (+0.91) |
| **BAIM** (Ours) | **80.16** ± 0.04 (**+0.27**) | **80.13** ± 0.03 (**+0.89**) | **80.18** ± 0.04 (**+0.23**) | **80.02** ± 0.03 (**+0.12**) | **80.36** ± 0.12 (**+1.06**) |

Table 2: Performance comparison (mean ± std) of KT backbones with different item representation methods. Values in parentheses denote absolute improvements over the Default setting; The best-performing method in terms of mean AUC is highlighted in **bold**.

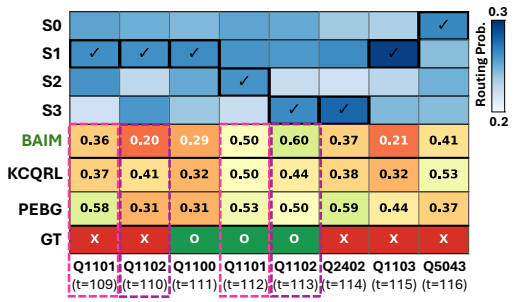



Figure 4: Visualization of BAIM's stage-level dynamic routing behavior and its impact on prediction outcomes under repeated interactions. **Rows 0–3**: routing weights over problem-solving stages; **Rows 4–6**: prediction outcomes of BAIM and baseline methods; **Row 7**: ground-truth (GT) student performances.

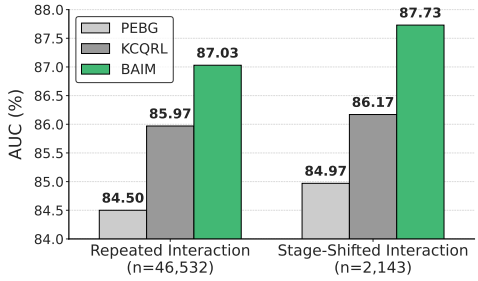



Figure 5: AUC comparison across repeated student interactions. Adaptive routing enables procedural focus shifts in BAIM.

representation methods, we analyze a sequence of learner-item interactions from the XES3G5M dataset using a sparseKT model trained on fold 0. Figure 4 visualizes BAIM's routing probabilities across problem-solving stages over time, alongside prediction outcomes from baseline methods. BAIM dynamically adjusts its routing focus to emphasize different solution stages as the learner's interaction context evolves, reflecting changes in procedural demand. This adaptive behavior is particularly evident for items Q1101 and Q1102, which are reattempted multiple times by the same learner. In contrast, baseline methods rely on fixed item representations, which may limit their adaptability and potentially account for their inaccurate predictive performance.

Quantitative results in Figure 5 further support the effectiveness of adaptive routing. Among 46,532 repeated interactions, BAIM changes its routing decision in 2,143 cases (stage-shifted subset) where adaptive behavior is explicitly triggered. BAIM outperforms the strongest baseline by 1.06 AUC on all repeated interactions, with the margin increasing to 1.56 AUC on the stage-shifted subset, highlighting the effectiveness of dynamically adapting stage-wise item representations.

## 7 Analysis

**Impact of Routing Strategy** We compare an adaptive routing strategy in BAIM with fixed aggregation strategies, including (i) selecting a single solution stage (Stage 0–3) and (ii) holistic pooling over the full solution trajectory without stage decomposition. Figure 7 shows that our adaptive routing strategy outperforms all single-stage models across all KT backbones, indicating that the most informative stage varies with the problem characteristics. Compared with holistic pooling, adaptive routing also yields consistently better performance, supporting the importance of explicit stage decomposition rather than relying on a single global representation.

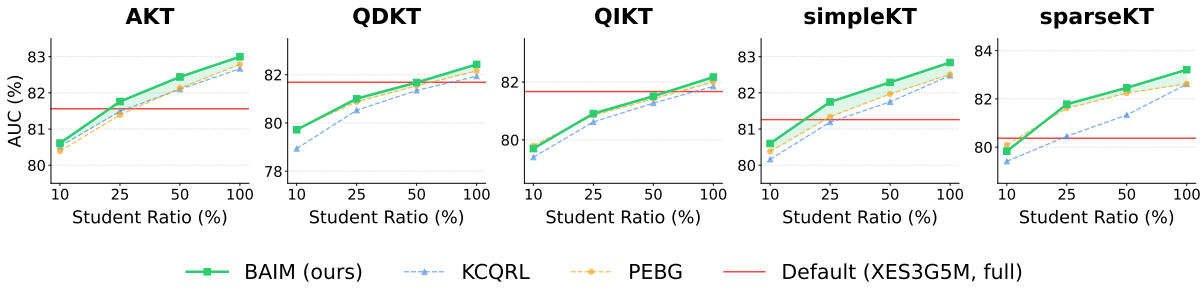



Figure 6: AUC (%, mean) of five KT backbones trained with varying numbers of students on XES3G5M. The red horizontal line indicates the performance of the Default representation trained on the full dataset (100%).

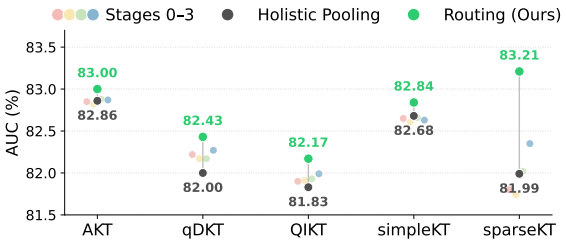



Figure 7: Performance comparison of aggregation strategies across multiple KT backbones on XES3G5M, highlighting the consistent advantage of BAIM's routing-based aggregation over fixed schemes.

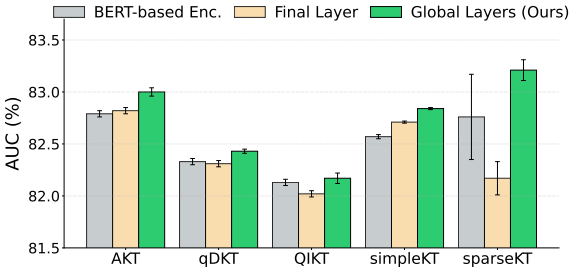



Figure 8: Impact of different representation extraction strategies. Leveraging the embedding trajectory of the solver RLM yields more effective item representations.

**Impact of Representation Extraction Strategies** We investigate the effect of different representation extraction strategies from the RLM solution process on downstream KT performance. All components of BAIM are kept fixed, and we vary only the aggregation strategy of hidden states across RLM layers or, alternatively, over generated solution texts to form stage-wise solution representations. Figure 8 demonstrates that leveraging information from the full depth of the RLM consistently outperforms representations derived from a single layer or sentence-level encoding across all KT backbones. Specifically, global layer pooling, which aggregates representations across the entire embedding trajectory ($l = [0, 64]$), achieves the strongest performance, outperforming both the final-layer ($l = 64$) and BERT-based encoding. In contrast, relying solely on the final layer leads to lower performance, suggesting that single-layer representations struggle to capture the diverse procedural and semantic information distributed throughout the depth of the RLM, consistent with recent findings on the benefits of intermediate and aggregated layer representations (Skean et al., 2025).

**Sample Efficiency** To analyze BAIM's robustness under limited training data, we vary the number of training students by randomly subsampling the XES3G5M dataset at ratios of 10%, 25%, 50%, and 100%. All models are trained from scratch on each subset and evaluated on the same test set. As detailed in Figure 6, BAIM achieves strong performance even with substantially fewer training students. Notably, for AKT, simpleKT, and sparseKT, BAIM trained with only 25% of the students already outperforms the `Default` model trained on the full dataset, demonstrating pronounced sample efficiency. Starting from the 25% ratio, BAIM consistently surpasses other item representation methods across all backbone architectures.

## 8 Conclusion

We propose BAIM, a behavior-aware item modeling framework for KT that represents items via structured problem-solving. By deriving stage-level representations from embedding trajectories of an RLM, BAIM captures rich procedural signals without auxiliary network pre-training. Moreover, BAIM adaptively integrates these representations via context-conditioned routing, allowing different problem-solving stages to be emphasized based on both procedural solution dynamics and learner histories. We demonstrate the effectiveness of BAIM across five representative KT backbones and two

real-world math learning datasets, where it consistently improves prediction performance, especially under repeated learner–item interactions.

## Acknowledgments

This research was supported by the Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2025 (Project Name: Development of an AI-Based Korean Diagnostic System for Efficient Korean Speaking Learning by Foreigners, Project Number: RS-2025-02413038, Contribution Rate: 45%); by the IITP(Institute of Information & Communications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT) (IITP-2026-RS-2024-00437866, Contribution Rate: 45%); and by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2019-II191906, Artificial Intelligence Graduate School Program(POSTECH), Contribution Rate: 10%).

We sincerely thank Deokhyung Kang and Chiyeong Heo for valuable discussions.

## Limitations

While BAIM effectively captures procedural item representations and consistently improves KT performance across diverse settings, several limitations remain. First, BAIM is designed and evaluated primarily in the context of mathematical problem solving. Extending the framework to other educational domains, such as programming tasks or second language learning, may require domain-specific adaptations to the stage formulation and procedural modeling. Second, our empirical evaluation is restricted to KT benchmarks that provide publicly available item content. As a result, we do not include several large and widely used datasets, such as ASSISTments, where item metadata are not publicly released, limiting direct comparison on these benchmarks.

## Ethical Statement

This work investigates item representation learning and does not involve ethical issues. All experiments are conducted using publicly accessible datasets. Specifically, the XES3G5M dataset and pyKT library are used under the MIT License, and the NIPS34 dataset is utilized in accordance with its official Terms of Service. The use of these datasets is strictly limited to academic research, which is consistent with their intended purpose and access conditions. GitHub Copilot was used to assist with code generation, and ChatGPT was used to support writing and language refinement. All research contributions are solely attributable to the authors.

## References

John R Anderson. 1996. Act: A simple theory of complex cognition. *American psychologist*, 51(4):355.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*, volume 4. Springer.

Jiahao Chen, Zitao Liu, Shuyan Huang, Qiongqiong Liu, and Weiqi Luo. 2023. Improving interpretability of deep sequential knowledge tracing models with question-centric cognitive representations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 14196–14204.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arxiv 2014. *arXiv preprint arXiv:1412.3555*, 1412.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Aritra Ghosh, Neil Heffernan, and Andrew S Lan. 2020. Context-aware attentive knowledge tracing. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2330–2339.

Shuyan Huang, Zitao Liu, Xiangyu Zhao, Weiqi Luo, and Jian Weng. 2023. Towards robust knowledge tracing models via k-sparse attention. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 2441–2445.

Tao Huang, Xinjia Ou, Huali Yang, Shengze Hu, Jing Geng, Junjie Hu, and Zhuoran Xu. 2024. Remembering is not applying: Interpretable knowledge tracing for problem-solving processes. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 3151–3159.

Ekaterina Krivich, Danial Hooshyar, Gustav Šír, Yeongwook Yang, Merja Bauters, Raija Hämäläinen, and Tommi Kärkkäinen. 2025. A systematic review of deep knowledge tracing (2015-2025): Toward responsible ai for education.

Unggi Lee, Sungjun Yoon, Joon Seo Yun, Kyoungsoo Park, YoungHoon Jung, Damji Stratton, and Hyeoncheol Kim. 2024. Difficulty-focused contrastive learning for knowledge tracing with a large language model-based difficulty prediction. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4891–4900.

Yunfei Liu, Yang Yang, Xianyu Chen, Jian Shen, Haifeng Zhang, and Yong Yu. 2020. Improving knowledge tracing via pre-training question embeddings. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1577–1583. International Joint Conferences on Artificial Intelligence Organization. Main track.

Zitao Liu, Qiongqiong Liu, Jiahao Chen, Shuyan Huang, and Weiqi Luo. 2023a. simpleKT: A simple but tough-to-beat baseline for knowledge tracing. In *The Eleventh International Conference on Learning Representations*.

Zitao Liu, Qiongqiong Liu, Teng Guo, Jiahao Chen, Shuyan Huang, Xiangyu Zhao, Jiliang Tang, Weiqi Luo, and Jian Weng. 2023b. Xes3g5m: A knowledge tracing benchmark dataset with auxiliary information. *Advances in Neural Information Processing Systems*, 36:32958–32970.

Yilmazcan Ozyurt, Stefan Feuerriegel, and Mrinmaya Sachan. 2024. Automated knowledge concept annotation and question representation learning for knowledge tracing. *arXiv preprint arXiv:2410.01727*.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. 2015. Deep knowledge tracing. *Advances in neural information processing systems*, 28.

George Pólya. 1957. *How to solve it: A new aspect of mathematical method*, 2nd edition. Princeton university press, Princeton, NJ.

Alan H Schoenfeld. 2014. *Mathematical problem solving*. Elsevier.

Alan H Schoenfeld and Douglas J Herrmann. 1982. Problem perception and knowledge structure in expert and novice mathematical problem solvers. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8(5):484.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, and 1 others. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. Layer by layer: Uncovering hidden representations in language models. In *Forty-second International Conference on Machine Learning*.

Xiangyu Song, Jianxin Li, Qi Lei, Wei Zhao, Yunliang Chen, and Ajmal Mian. 2022. Bi-clkt: Bi-graph contrastive learning based knowledge tracing. *Knowledge-Based Systems*, 241:108274.

Shashank Sonkar, Andrew E Waters, Andrew S Lan, Phillip J Grimaldi, and Richard G Baraniuk. 2020. qdkt: Question-centric deep knowledge tracing. *arXiv preprint arXiv:2005.12442*.

John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.

Yixuan Tang and Yi Yang. 2024. Pooling and attention: What are effective designs for llm-based embedding models? *arXiv preprint arXiv:2409.02727*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, and 1 others. 2025a. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

Wentao Wang, Huifang Ma, Yan Zhao, and Zhixin Li. 2024a. Pre-training question embeddings for improving knowledge tracing with self-supervised bi-graph co-contrastive learning. *ACM Trans. Knowl. Discov. Data*.

| Notation | Shape |
|---|---|
| $D_{\text{input}}$ | 768 |
| $D_{\text{kt}}$ | 256 |
| $D_{\text{history}}$ | 64 |
| $f_{\text{proj}}(\cdot)$ | Linear($4D_{\text{input}} \to D_{\text{kt}}$) + ReLU + Dropout |
| $\mathbf{W}_{\text{in}}$ | $\in \mathbb{R}^{D_{\text{history}} \times (2D_{\text{kt}})}$ |
| $\mathbf{W}_{\text{gate}}$ | $\in \mathbb{R}^{4 \times (D_{\text{kt}} + D_{\text{history}})}$ |
| $\mathbf{b}_{\text{gate}}$ | $\in \mathbb{R}^4$ |
| $\mathbf{r}_{t-1}$ | $\in \mathbb{R}^{D_{\text{kt}}}$, $r_{t-1} \in \{0, 1\}$ |
| $\mathbf{m}_t$ | $\in \mathbb{R}^{D_{\text{history}}}$ |
| $\text{MLP}_p(\cdot)$ | Linear($D_{\text{input}} \to D_{\text{input}}$) + ReLU + Dropout + Linear($D_{\text{input}} \to D_{\text{kt}}$) |

Table 3: Notations and tensor shapes used in the BAIM architecture.


Wentao Wang, Huifang Ma, Yan Zhao, Fanyi Yang, and Liang Chang. 2022. Perm: Pre-training question embeddings via relation map for improving knowledge tracing. In *International Conference on Database Systems for Advanced Applications*, pages 281–288. Springer.

Yiming Wang, Pei Zhang, Baosong Yang, Derek Wong, Zhuosheng Zhang, and Rui Wang. 2024b. Embedding trajectory for out-of-distribution detection in mathematical reasoning. *Advances in Neural Information Processing Systems*, 37:42965–42999.

Yiming Wang, Pei Zhang, Baosong Yang, Derek F. Wong, and Rui Wang. 2025b. Latent space chain-of-embedding enables output-free LLM self-evaluation. In *The Thirteenth International Conference on Learning Representations*.

Zichao Wang, Angus Lamb, Evgeny Saveliev, Pashmina Cameron, Yordan Zaykov, José Miguel Hernández-Lobato, Richard E. Turner, Richard G. Baraniuk, Craig Barton, Simon Peyton Jones, Simon Woodhead, and Cheng Zhang. 2020. Instructions and guide for diagnostic questions: The NeurIPS 2020 education challenge. In *NeurIPS 2020 Competition and Demo Track*, volume 133, pages 151–169. PMLR.

Bihan Xu, Zhenya Huang, Jiayu Liu, Shuanghong Shen, Qi Liu, Enhong Chen, Jinze Wu, and Shijin Wang. 2023. Learning behavior-oriented knowledge tracing. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pages 2789–2800.


## A Architectural Details of BAIM

This section describes the implementation details of the neural components in the BAIM framework, including the MLPs, recurrent modules, and routing mechanisms. A summary of all notations and tensor shapes used throughout the architecture is provided in Table 3.

**Procedural Solution Representation.** For each item $I_t$, the stage-aware item embeddings $\{\mathbf{h}'_{t,p}\}_{p=0}^{3}$ are concatenated and passed through a projection network $f_{\text{proj}}(\cdot)$ to produce a solution representation $\mathbf{s}_t \in \mathbb{R}^{D_{\text{kt}}}$. The projection network consists of a linear transformation Linear($4D_{\text{input}} \to D_{\text{kt}}$) followed by ReLU activation and dropout.

**Learner Interaction Context Encoder.** At each time step $t$, the GRU updates the latent context by taking as input the concatenation of a stage-aware solution representation derived from the previous item and the previous response embedding $\mathbf{r}_{t-1}$. The response embedding $\mathbf{r}_{t-1}$ is obtained from the binary response $r_{t-1} \in \{0, 1\}$ via a learnable lookup table. The concatenated input is projected into the context space via $\mathbf{W}_{\text{in}} \in \mathbb{R}^{D_{\text{history}} \times (2D_{\text{kt}})}$ before being fed into the GRU.

**Routing Gate Network.** During training, Gaussian noise is injected into the routing logits to encourage exploration and stabilize routing behavior, following common practice in sparse mixture-of-experts routing (Shazeer et al., 2017):

$$\tilde{\boldsymbol{\alpha}}_t = \boldsymbol{\alpha}_t + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}\big(\mathbf{0}, (1/4)^2 \mathbf{I}\big)\,. \tag{11}$$

Routing decisions are made via a Top-1 selection based on the noisy logits $\tilde{\boldsymbol{\alpha}}_t$, while deterministic routing using $\boldsymbol{\alpha}_t$ is employed at inference time.

**Stage-Specific Expert Networks.** Each procedural reasoning stage $p \in \{0, 1, 2, 3\}$ is associated with a lightweight expert MLP with an identical architecture but independent parameters. Concretely, each expert maps its corresponding stage-level embedding from $\mathbb{R}^{D_{\text{input}}}$ to $\mathbb{R}^{D_{\text{kt}}}$ via a two-layer feed-forward network consisting of a linear transformation ($D_{\text{input}} \to D_{\text{input}}$), followed by ReLU activation and dropout, and a second linear projection ($D_{\text{input}} \to D_{\text{kt}}$). All expert outputs are computed in parallel, and a Top-1 routing strategy is applied such that only the selected expert output contributes to the final representation. A shared layer normalization is applied to the selected output, yielding the final item representation $\mathbf{I}_t \in \mathbb{R}^{D_{\text{kt}}}$, which is subsequently passed to the KT backbone for response prediction.

## B Details on Baseline Reproductions

**Default Setting.** In the **Default** setting, item and knowledge-component (KC) embeddings are randomly initialized and trained end-to-end together

with each KT backbone. For backbones that explicitly model KC-level representations, including AKT, SimpleKT, and SparseKT, training is performed at the KC level. At inference time, item-level predictions are obtained via a late fusion strategy that aggregates KC-level outputs associated with each item. This design follows the standard usage of these backbones and ensures that all models operate under their intended training paradigms.

**PEBG Reproduction.** We reproduce PEBG following the official implementation with minimal dataset-specific adjustments. While most architectural details and preprocessing steps strictly follow the original work, we adjust the model scale to match our experimental environment. Specifically, we set the embedding and hidden dimensions to $768$ (originally $64$ and $128$, respectively) and apply a dropout rate of $0.3$ to prevent overfitting. For items absent from the training set, we assign default attributes ($ms = 0.0$, $p\_correct = 0.5$) to maintain graph connectivity.

**KCQRL Reproduction.** We reproduce KCQRL using the released code and data under the original contrastive framework, yielding $768$-dimensional item embeddings. For experiments using pretrained item embeddings from either KCQRL or PEBG, we employ item-level KT backbone variants consistent with the KCQRL architecture to enable uniform integration and evaluation.

## C Architecture-Specific Considerations for Item Representation

We integrate BAIM into existing KT backbones by modifying only the item representation module, while preserving each model's original sequence modeling and prediction mechanisms. Based on how item representations are consumed within each backbone, we categorize the models into three groups.

**Group A: AKT and qDKT.** AKT and qDKT require separate embeddings for the current item and historical item–response interactions. Accordingly, BAIM replaces the original item and interaction embedding modules with learner-conditioned item representations. Specifically, BAIM produces a learner-conditioned item representation $\mathbf{I}_t \in \mathbb{R}^{D_{\text{kt}}}$, which serves as the shared source representation for constructing item–response embeddings. Following the original designs of AKT and qDKT, response-aware interaction embeddings are obtained by applying response-specific linear transformations to $\mathbf{I}_t$, where separate projection matrices are used for correct and incorrect responses, respectively. In addition, a dedicated linear projection is applied to $\mathbf{I}_t$ to construct the query embedding for the current item. All subsequent attention, sequence modeling, and prediction components are preserved exactly as in the original implementations.

**Group B: QIKT.** QIKT maintains an explicit separation between item embeddings and KC embeddings. To respect this design, BAIM is integrated only at the item level. The learner-conditioned item representation produced by BAIM replaces the original item representation, while the concept embedding module and item–concept fusion mechanism remain unchanged. Thus, BAIM does not introduce or modify any concept-level representations in QIKT.

**Group C: simpleKT and sparseKT.** simpleKT and sparseKT employ single-stream Transformer architectures in which item embeddings are directly used as query representations. For these models, BAIM replaces the original static item embeddings with learner-conditioned item representations projected to $D_{\text{kt}}$. Following the original designs of simpleKT and sparseKT, response information is incorporated by directly combining the learner-conditioned item representation with the response embedding, rather than through response-specific linear projections. In sparseKT, the original sparse attention mechanism and its hyperparameters are fully preserved; BAIM only affects the input embedding supplied to the Transformer. This ensures that the sparsification behavior remains identical to the original implementation.

## D Data Preprocessing Details

In this section, we provide detailed descriptions of the preprocessing pipelines for the two benchmark datasets used in our study to ensure reproducibility and transparency in our procedural solution extraction process.

**XES3G5M Metadata Translation.** The XES3G5M dataset was collected from a Chinese online learning platform, where all question texts and analytical analyses are provided in Chinese. Although the underlying solver RLM supports Chinese, we translate the metadata into

| Method \ Backbone | Knowledge Tracing Backbone Architecture | | | | |
|---|---|---|---|---|---|
| | AKT | qDKT | QIKT | simpleKT | sparseKT |
| **XES3G5M** | | | | | |
| InternVL-3.5-8B | 82.95 ± 0.04 | 82.39 ± 0.02 | 82.15 ± 0.02 | 82.81 ± 0.03 | 83.17 ± 0.09 |
| Qwen3-VL-8B-Thinking | 82.95 ± 0.05 | 82.39 ± 0.02 | 82.14 ± 0.03 | **82.84** ± 0.01 | **83.22** ± 0.08 |
| Qwen3-VL-32B-Thinking | **83.00** ± 0.04 | **82.43** ± 0.02 | **82.17** ± 0.05 | **82.84** ± 0.01 | 83.21 ± 0.10 |
| **NIPS34** | | | | | |
| InternVL-3.5-8B | 80.12 ± 0.06 | 80.11 ± 0.01 | 80.16 ± 0.05 | 79.99 ± 0.03 | 80.34 ± 0.09 |
| Qwen3-VL-8B-Thinking | 80.14 ± 0.03 | **80.15** ± 0.03 | **80.18** ± 0.01 | **80.02** ± 0.02 | **80.45** ± 0.07 |
| Qwen3-VL-32B-Thinking | **80.16** ± 0.04 | 80.13 ± 0.03 | **80.18** ± 0.04 | **80.02** ± 0.03 | 80.36 ± 0.12 |

Table 4: Performance comparison (mean ± std) of BAIM-equipped KT backbones using different solver RLMs. The best-performing solver in terms of mean AUC is highlighted in **bold**.

English to standardize the working language of our analysis pipeline and to enable more reliable human inspection, error analysis, and qualitative evaluation. Translation is performed using the GPT-5-nano model (Singh et al., 2025), with particular care taken to preserve mathematical notation and the logical structure of the original analyses.

During the translation process, we identified indexing issues, particularly cases where image file names were included in the option fields. These entries are treated as annotation noise originating from the source dataset. The manually corrected metadata is used in all our experiments, and the finalized version is fully available in our public repository.

**NIPS34 Image-based Metadata Generation** Since the NIPS34 provides only question images without structured text or analysis, we generate metadata using a **Gemini-2.5-Pro** (Comanici et al., 2025). Given an input image, the model jointly performs visual understanding and text generation to extract the question content, multiple-choice options, a concise analytical explanation, and the correct answer. The full prompt used for metadata generation is shown in Figure 9.

The generated analytical explanations serve as reference material for downstream reasoning extraction, enabling the solver module to derive stage-wise problem-solving trajectories. This preprocessing step allows BAIM to be applied to image-based educational datasets that do not natively provide textual problem descriptions or procedural solution traces.

**Interaction Filtering and Splitting.** Following the standard protocol of the pyKT library, all datasets were divided into training, validation, and test sets using the default splitting ratios to facilitate fair comparison with existing baselines.

| Stage | Mean | Std Dev | Min | Max |
|---|---|---|---|---|
| ***Qwen3-VL-32B-Thinking*** | | | | |
| **Thinking Process** | **1,191.78** | **1,107.05** | **206** | **11,704** |
| Stage 1: Understand | 50.05 | 15.29 | 16 | 149 |
| Stage 2: Plan | 43.39 | 14.23 | 9 | 140 |
| Stage 3: Carry Out | 76.80 | 40.49 | 7 | 375 |
| Stage 4: Look Back | 36.52 | 25.20 | 9 | 1,773 |
| *Total Sequence* | 1,423.48 | 1,128.02 | 340 | 11,999 |
| ***Qwen3-VL-8B-Thinking*** | | | | |
| **Thinking Process** | **2,064.72** | **2,245.83** | **214** | **17,655** |
| Stage 1: Understand | 56.28 | 17.87 | 16 | 198 |
| Stage 2: Plan | 50.09 | 22.56 | 12 | 348 |
| Stage 3: Carry Out | 96.15 | 63.31 | 9 | 892 |
| Stage 4: Look Back | 50.66 | 24.12 | 10 | 275 |
| *Total Sequence* | 2,344.07 | 2,286.52 | 354 | 17,941 |
| ***InternVL-3.5-8B*** | | | | |
| **Thinking Process** | **1,232.83** | **1,569.69** | **46** | **11,628** |
| Stage 1: Understand | 39.51 | 13.81 | 9 | 161 |
| Stage 2: Plan | 34.54 | 15.90 | 8 | 273 |
| Stage 3: Carry Out | 83.24 | 55.55 | 10 | 1,786 |
| Stage 4: Look Back | 30.56 | 17.13 | 4 | 689 |
| *Total Sequence* | 1,420.69 | 1,589.12 | 125 | 11,855 |

Table 5: Token usage statistics for solver RLM generation across model variants on the XES3G5M.

## E Hardware Usage

For our experiments, we used a single NVIDIA GeForce RTX 3090 GPU for training the KT models, and two NVIDIA L40S GPUs for RLM inference.

## F RLM-Family Analysis

Table 4 shows that BAIM consistently improves performance across all evaluated KT backbones on XES3G5M and NIPS34. Across different solver families, the overall performance remains compa-

| $\lambda$ | AUC | Stage 0 | Stage 1 | Stage 2 | Stage 3 |
|---|---|---|---|---|---|
| 0 | $83.18 \pm 0.06$ | 0.48 | 0.44 | 0.08 | 0.01 |
| 0.01 | $\mathbf{83.21 \pm 0.10}$ | 0.24 | 0.29 | 0.23 | 0.24 |
| 0.1 | $83.16 \pm 0.19$ | 0.28 | 0.23 | 0.31 | 0.18 |

Table 6: Effect of the load-balancing coefficient $\lambda$ on XES3G5M with sparseKT. We report test AUC and aggregated Top-1 routing proportions over the four procedural stages.

rable, indicating that BAIM is robust to the choice of solver model. In particular, the performance gap between Qwen3-VL-8B-Thinking and Qwen3-VL-32B-Thinking is marginal across most backbones, suggesting that the procedural information extracted by the solver does not strongly depend on model scale once a sufficient capacity threshold is reached.

In practice, we adopt Qwen3-VL-32B-Thinking in the main experiments because Qwen3-VL-8B-Thinking more frequently exhibits overthinking, producing unnecessarily long reasoning traces. While downstream performance remains comparable, this behavior increases preprocessing cost and reduces generation efficiency. Detailed token usage statistics are reported in Table 5.

## G Effect of the Load-Balancing Loss

We analyze the effect of the load-balancing regularizer by varying its coefficient $\lambda \in \{0, 0.01, 0.1\}$ on XES3G5M with the sparseKT backbone. As shown in Table 6, the overall AUC is similar across settings, with $\lambda = 0.01$ achieving the best performance. However, the routing behavior differs substantially. Without load balancing ($\lambda = 0$), Top-1 routing collapses to a small subset of stages, with about $92\%$ of samples assigned to Stage 0 or Stage 1. In contrast, $\lambda = 0.01$ yields a much more balanced routing distribution across all four stages, while preserving the best AUC. A larger value, $\lambda = 0.1$, also mitigates collapse but gives slightly lower performance. These results suggest that an appropriate choice of $\lambda$ can effectively prevent stage collapse while also providing a modest improvement in AUC.

## H Solver RLM Inference

**Decoding Hyperparameters.** Because we observed overthinking behavior in Qwen3-VL-8B-Thinking, we set `max_tokens`=18000 for Qwen3-VL-8B-Thinking, and `max_tokens`=12000 for Qwen3-VL-32B-Thinking and InternVL-3.5-8B (Wang et al., 2025a). Other hyperparameters are fixed to temperature = `0.7`, top-$p$ = `0.9`, and repetition penalty = `1.1`.

**Prompting Setup.** We use a fixed prompt to elicit Polya-style four-stage reasoning in JSON format. The full prompt is shown in Figure 10.

**Robustness to RLM Reasoning Errors.** To evaluate the reliability of the solver RLM, we manually inspected $1,000$ randomly sampled outputs of Qwen3-VL-32B-Thinking on the XES3G5M. We identified that only $1.3\%$ of the cases exhibited logical inconsistencies or incorrect final answers, typically occurring when the source metadata was highly ambiguous or the reference analysis was overly concise.

## Prompt

**SYSTEM PROMPT**

You are a math education expert. Your task is to read the provided question image and extract its content into a structured JSON representation suitable for downstream reasoning.

**Output Format Rules**

- Output **one valid JSON object only** (no additional text).
- The JSON must contain exactly the following fields:
  - `"question"`: Concise textual description.
  - `"options"`: Dict with keys `"A"`, `"B"`, `"C"`, `"D"`.
  - `"analysis"`: Brief explanation (3–4 sentences).
  - `"answer"`: Correct option key.

**Content Constraints**

- Use LaTeX for all mathematical expressions.
- Reference figures as `question_{id}-image_0`.
- Ensure JSON is syntactically valid.

---

**INSTRUCTION**

Process the given question image and generate structured metadata that faithfully captures the problem statement, solution logic, and correct answer.

**IMPORTANT**

- Do not include extraneous commentary.
- Follow the specified JSON schema strictly.

Image: `{question_image}`

Figure 9: Prompt used to generate structured metadata from question images using Gemini-2.5-Pro.

## Prompt

**SHARED SYSTEM PROMPT**
You are a math student solving problems using **Polya's 4-Stage Process**. You must explain your reasoning **in English only**.
**Output Format Rules**

- Immediately after `</think>`, output **one JSON object only** (no extra text).
- The JSON must have exactly these 4 keys:
  1. `"understand"`: Identify knowns, unknowns, and conditions.
  2. `"plan"`: State the strategy, formulas, or theorems to be used.
  3. `"carry_out"`: Execute the plan step-by-step with calculations.
  4. `"look_back"`: State the final answer and verify if it makes sense.

**Content Constraints**

- Keep sentences concise and logical.
- Ensure the JSON is valid and parseable.

**Example Output**

```
{
  "understand": "Given a circle with radius 5. Need to find the area.",
  "plan": "Use the area formula A = pi * r^2. Substitute r=5.",
  "carry_out": "A = pi * 5^2 = 25 * pi. Approx 78.5.",
  "look_back": "The result 25pi is positive and reasonable. Final Answer: 25pi"
}
```

---

**INSTRUCTION**
Simulate a **high-performing student**. Solve the following problem by strictly following Polya's 4 stages correctly.

**IMPORTANT**: You are provided with a reference **Analysis**.

- Do not just copy the Analysis.
- **Reconstruct** the reasoning path based on the Analysis to ensure your steps and calculations are 100% accurate.
- **understand**: Extract key information from the Question.
- **plan**: Formulate a strategy consistent with the provided Analysis.
- **carry_out**: Perform precise calculations step-by-step as outlined in the Analysis.
- **look_back**: Verify the result matches the Analysis's conclusion.

Question: `{question}`
Analysis: `{analysis}`

Figure 10: System prompt used to elicit Polya-style four-stage reasoning trajectories from the solver RLM.