Scalable and Equitable Math Problem Solving Strategy Prediction in Big Educational Data

Anup Shakya
University of Memphis
ashakya@memphis.edu

Vasile Rus
University of Memphis
vrus@memphis.edu

Deepak Venugopal
University of Memphis
dvngopal@memphis.edu

ABSTRACT

Understanding a student's problem-solving strategy can have a significant impact on effective math learning using Intelligent Tutoring Systems (ITSs) and Adaptive Instructional Systems (AISs). For instance, the ITS/AIS can better personalize itself to correct specific misconceptions that are indicated by incorrect strategies, specific problems can be designed to improve strategies and frustration can be minimized by adapting to a student's natural way of thinking rather than trying to fit a standard strategy for all. While it may be possible for human experts to identify strategies manually in classroom settings with sufficient student interaction, it is not possible to scale this up to big data. Therefore, we leverage advances in Machine Learning and AI methods to perform scalable strategy prediction that is also fair to students at all skill levels. Specifically, we develop an embedding called MVec where we learn a representation based on the mastery of students. We then cluster these embeddings with a non-parametric clustering method where we progressively learn clusters such that we group together instances that have approximately symmetrical strategies. The strategy prediction model is trained on instances sampled from these clusters. This ensures that we train the model over diverse strategies and also that strategies from a particular group do not bias the DNN model, thus allowing it to optimize its parameters over all groups. Using real world large-scale student interaction datasets from MATHia, we implement our approach using transformers and Node2Vec for learning the mastery embeddings and LSTMs for predicting strategies. We show that our approach can scale up to achieve high accuracy by training on a small sample of a large dataset and also has predictive equality, i.e., it can predict strategies equally well for learners at diverse skill levels.

Keywords

Intelligent Tutoring Systems, Strategy Prediction, Equity, Representation Learning, Skill Mastery, Non-parametric Clustering, Fairness, Transformers, LSTM, Symmetry

A. Shakya, V. Rus, and D. Venugopal. Scalable and equitable math problem solving strategy prediction in big educational data. In M. Feng, T. Käser, and P. Talukdar, editors, *Proceedings of the 16th International Conference on Educational Data Mining*, pages 137–148, Bengaluru, India, July 2023. International Educational Data Mining Society.

https://doi.org/10.5281/zenodo.8115669

1. INTRODUCTION

The recent pandemic has spurred a remarkable growth in virtual learning and with it, the necessity to develop learning technologies that are effective even in the absence of face-to-face instruction. To this end, Intelligent Tutoring Systems (ITSs) [25] and more broadly Adaptive Instructional systems (AISs) will play a key role in education since they can scale up personalized instruction to large and diverse student populations. However, to adapt to a student, an AIS should be able to understand the student's thinking process which can be challenging. For instance, if we consider math learning, students can solve the same problem using several different approaches or *strategies*. Understanding these strategies can help an ITS/AIS adapt more effectively [22]. For example, the type of strategy can reveal the expertise/knowledge of a student in a topic, incorrect strategies that indicate misconceptions can be corrected by the ITS, the student can be trained to change strategy based on the problem context, and students may be less frustrated if the ITS guides them towards strategies that are more naturally aligned to their thinking.

In math problem solving, a *strategy* is a sequence of actions/steps that the student performs to solve a problem. An example of 3 different strategies is shown in Fig. 1. Human tutors can recognize different strategies followed by students and utilize these in one-on-one instruction. For instance, if a student is a visual learner, then they can teach the student to solve problems through visual aids, or if the student prefers an analytical approach to solve the same problem, then they can modify their teaching accordingly. However, adapting this approach for ITSs is challenging, particularly since identifying problem-solving strategies through computational methods is a complex problem. Specifically, there may be several strategies that are similar/symmetric without being completely identical. An example is illustrated in Fig. 1 to show similar and dissimilar strategies. As shown here, 2 of the 3 strategies are not exactly identical but implement the same idea and are thus symmetrical. The third strategy is quite different and asymmetrical to the first two strategies. Further, there may be several strategies that may not be conventional approaches to problem-solving but are indicative of unique ways in which students think about problems. Thus, if we identify a new strategy based on matching them with a set of previously known strategies, this approach may not be very effective when we want to scale up to big educational data. While there have been several approaches to detect strategies including using model

tracing [4] or sequence mining [34] methods, newer advances in deep neural networks (DNNs) can learn much more complex representations from large-scale data. Thus, leveraging such DNNs, we predict *novel* strategies more effectively.

Our goal in this paper is to develop a scalable and equitable model to predict strategies in math learning. Specifically, though DNN models are highly effective, they may tend to produce biased results. For instance, since most DNNs have a loss function that optimizes the overall loss, depending on the data distribution used during training, their results may be unfair to some sub-groups in the data. In our context, we want to avoid the model being unfairly biased where it can only identify strategies for certain student sub-groups. Specifically, we want to avoid *disparate mistreatment* [35] where the model accuracy is significantly different for different types of learners. In particular, learners may have disparity in their mastery or skill level which will influence their choice of strategy for a problem. For example, in Fig. 1, the third strategy shown in the figure is more sophisticated than the other two and the student who applies this strategy is likely to have greater mastery in the topic. Therefore, we want to ensure that our model can predict strategies equally well for learners at all skill levels. To do this, we use a sampling approach, where instead of training the DNN over the full dataset (which may contain biases), we modify the underlying data distribution. Specifically, we sample the data such that sub-groups in the data are equally well-represented. Thus, when the DNN is trained over these samples instead of the full dataset, the DNN is forced to optimize its loss over all sub-groups. In general, sampling is a well-known approach used to scale up complex DNNs while training the model from large datasets [6]. Further, it has been shown that in some cases using too much data can lead to poor generalization [17]. In our case, a naive sampling approach where we sample students uniformly at random and train over strategies used by the sampled students will certainly be biased towards the skill level of the majority group and does not account for inequalities in skill levels. Therefore, here, we develop an iterative non-parametric clustering method where we cluster the data into groups where each group corresponds to strategies corresponding to similar skills levels. Further, since strategies themselves are hard to compare exactly, we develop an approach where we use *approximate symmetries* to group strategies. We then train a DNN to predict a strategy by sampling from these diverse groups.

We implement our approach using the *DP-Means* Hierarchical Dirichlet Process framework [9] to jointly cluster students and problems. Specifically, we project students (and problems) into an embedding space that we term *MVec* (Mastery Vectorization). To do this, we represent relationships between symbolic objects (students, problems, and concepts used in strategy) as a graphical structure. We then learn dense vectors using an embedding approach called *Node2Vec* [5] that assigns similar embeddings to nodes that have similar neighborhoods. We add mastery over concepts used in the strategy as weights in the graph estimated from a transformer model with attentions [31]. Thus, students with mastery over similar concepts in their strategies are assigned similar embeddings. We optimize the clusters incrementally where in each step, we adaptively change a penalty parameter based on the symmetries encoded by the clusters in the previous step. To quantify approximate symmetries, we develop a strategy alignment procedure with *positional encodings* [28]. Once the clusters converge, we sample training instances from the clusters and train a Long Short Term Memory (LSTM) model that predicts strategies.

We evaluate our approach on two datasets from MATHia, a commercial AIS widely used for math learning in schools. The data is available through the PSLC datashop [30]. The datasets are both large datasets that consist of millions of data instances (an instance is a student-problem pair and has multiple interactions in the dataset). Our results confirm that using our approach, we can sample a substantially smaller set of instances from the big dataset which we can use to train the strategy prediction model efficiently and achieve high accuracy in strategy prediction for students at diverse levels of mastery.

2. BACKGROUND

2.1 Related Work

Ritter et al. [22] provide a comprehensive survey on different approaches used to identify student strategies. Well-known approaches include the use of model tracing-based methods [4] to identify strategies. In such cases, strategies may be pre-specified and the tutor can recognize correct and incorrect strategies. Model-tracing-based methods have also been adapted to recognize new strategies [21]. Sequence learning approaches have been used in Open-Ended Learning Environments such as Betty's brain [11]. In [34], sequence pattern mining was applied to a MOOCs platform to analyze activity sequences of learners. For conversational tutors, natural language conversation interactions between tutors and students were mapped into a taxonomy of higher-level pedagogical concepts (e.g. scaffolding) by education experts [15]. These concepts can also be seen as a form of strategy and models have been developed to predict these concepts from conversational tutors [13, 26, 32]. Shakya et al. [27] developed an approach using importance sampling to sample data instances to scale up training of a strategy prediction model based on student interaction data from Mathia. Specifically, they formulated a Neuro-Symbolic AI model [33] where symbolic formulas were used in conjunction with a DNN to train the model. However, unlike our approach [27] has two fundamental limitations in identifying strategies. Particularly, their work does not use mastery to diversify the training samples which is important for equitable training. Further, it does not learn approximately symmetrical groups in a non-parametric manner. Thus, it cannot effectively group together symmetrical strategies which is necessary if we want to train the DNN from strategies that represent all such groups.

Mastery-based learning was proposed in the classic work by Bloom [1] to reduce achievement gaps between diverse students. The famous Bloom 2-sigma rule illustrates the benefits of such mastery-based learning. Ritter et al. [23] more recently provides a detailed insight into how mastery learning works in large-scale environments through their experiments on the MATHia platform. Knowledge tracing [4] is a well-known approach for inferring the *knowledge state* of students over KCs which indicates the degree of mastery over the KCs. More recently, deep knowledge tracing [20]

Solve linear equations: $5x - 2y = 4$ ①
$3x + 2y = 12$ ②

Strategy 1			Add ① and ②	Eliminate y	Collect x on left	Divide by coef. of x	find x	substitute x to find y
Strategy 2	Mul ① by 3	Mul ② by 5	Sub ① from ②	Eliminate x	Collect y on left	Divide by coef. of y	find y	substitute y to find x
Strategy 3			Formulate mat. mul. $AX = B$	Identify A, X and B	Compute A^{-1}	Compute $X = A^{-1}B$	find x and y	

Figure 1: Illustrating symmetries in strategies where similar colors indicate similar steps. Strategies 1 and 2 are similar in that they use the elimination method but are not identical. Strategy 3 uses the matrix method which indicates a higher level of sophistication in student mastery.

performed knowledge tracing using deep learning models. There is also a significant momentum in tackling the Knowledge Tracing problem in terms of graphs with the advent of GNNs [12, 16]. In [29], node-level and graph-level GCNs have been used to learn exercise-to-exercise and concept-to-concept relational sub-graphs adding to the semantic value of the representations. The natural phenomenon of learning, forgetting and dynamic changes to a student's mastery of knowledge concepts is formulated using gating-controlled mechanisms in [36]. Learning the pre-requisite structure of various associated skills has proven to be insightful to understand the problem-solving patterns [3, 19]. In [18], an attention-based model was proposed to predict correct answers but this was not used to predict strategies which is the focus of our work.

Our approach to using symmetries to make deep learning more scalable is inspired by the Geometric Deep Learning (GDL) [2] framework. Specifically, GDL is a formal framework used to understand the effectiveness of DNNs from the perspective of symmetries. Here, we ground GDL in the context of improving the effectiveness of DNNs in strategy prediction from big, diverse data. More generally, being selective about training instances has been shown to improve scalability and generalization [17]. Deep importance sampling [6] has the same underlying principle as our approach in that they propose to sample data to scale up training. However, unlike our approach they do not use symmetries as a basis for efficiently and equitably training the model. More recently, there has been work on improving fairness in DNNs by adaptively selecting batches during training to improve fairness measures such as minimizing gender disparity [24]. In principle, our approach also tries to achieve a similar goal in the context of educational data which is more challenging given that both mastery and strategies are complex variables.

2.2 Overview of Embedding Models

We use the well-known embedding model Node2Vec [5] to learn our mastery-based embedding MVec. Node2Vec is an embedding model for graphs and learns embeddings/dense vectors for nodes in the graph base on local neighborhoods. It is well-known to be a highly scalable approach for learning embeddings from large graphs. Node2Vec assigns similar vector representations for nodes with similar neighborhoods. Internally, it uses a skip-gram model called Word2Vec [14] to learn these representations. Word2Vec, which was originally developed for word embeddings, is used to predict neighboring nodes (also called context) from a given node. An autoencoder architecture is used in Word2Vec and the hidden layer learns the embedding. When neighborhoods are similar for two nodes, since their contexts are similar, the embedding learned for the two nodes will also be similar. Thus, Word2Vec projects the nodes into a continuous embedding space where similar/symmetrical nodes lie close to each other in the space.

2.3 DP-Means

DP-Means [10] is a non-parametric clustering algorithm that does not require us to specify of the number of clusters. The DP-Means Hard Gaussian Processes (HDP) clustering learns a 2-step hierarchy where *local clusters* for multiple datasets are learned at the lower level and these clusters are associated with *global clusters* at the higher level. Let x_{ij} denote the i-th instance of dataset j. The specific objective function of HDP is as follows.

$$\sum_{p=1}^{g} \sum_{x_{ij} \in \ell_p} ||x_{ij} - \mu_p||_2^2 + \lambda_\ell k + \lambda_g g \tag{1}$$

where ℓ_p is the p-th global cluster, k is the total number of local clusters, μ_p is the center of the p-th global cluster, g is the total number of global clusters, λ_ℓ is a local penalty that controls the formation of local clusters and λ_g is a global penalty that controls the formation of global clusters.

We can minimize the objective in Eq. (1) HDP clustering as follows. For each x_{ij}, we compute the distance to the current global cluster means. If the minimal distance exceeds $\lambda_\ell + \lambda_g$, we create a new local cluster for x_{ij} and a new global cluster ℓ_g associating it with the newly created local cluster. If the minimal distance is smaller than the sum of penalties, then we find the closest global cluster for x_{ij}, say $\ell_{g'}$. We then add x_{ij} to a local cluster that is already a part of $\ell_{g'}$. If no such local clusters exist, we create a new one for x_{ij} and associate it with $\ell_{g'}$. We then process the local clusters as follows. Let c denote a local cluster. We compute the global cluster whose mean is at a minimal distance, d' from c. Let the sum of distances of the points in the local cluster c to its cluster center be m. If d' is greater than the sum of the global cluster penalty and m, we create a new global cluster and assign c to this new global cluster. This algorithm converges to a locally optimal solution for Eq. (1) as shown in [10].

2.4 Positional Encodings

Positional encodings [31] are used to encode positional information in a sequence using a continuous vector space. Specifically, using sine and cosine functions that alternate with frequencies, we can represent positions in a sequence as follows. Let the position of the t-th item in the sequence be encoded by the d dimensional vector $\vec{p}_t$. The k-th dimension in $\vec{p}_t$ is computed as follows. If k is even, the value is equal to the sinusoidal function $sin(\omega_k.t)$ and if k is odd,

the value is equal to the cosine function $cos(\omega_k.t)$, where $\omega_k = 1/10000^{2k/d}$. The frequencies of the sine and cosine functions increase as k increases. Positional encodings are widely used to augment the latent representation learned by a deep network with positional information for sequence learning.

3. PROPOSED APPROACH

Since strategy is a generic term, we define it more precisely. Specifically, we consider strategies in the context of structured interaction between students and tutors. In this case, a student interacts with a tutor and solves a problem by sequentially solving the steps that lead to the final solution. Thus, we can think of a strategy as a sequence of actions the student takes among possible sequences in an action-space. Operationally, each step in the sequence is associated with a specific *knowledge component* (KC) [8] which is defined by domain experts and corresponds to the concept/knowledge required to solve that step. Thus, in our discussion, a strategy corresponds to a sequence of KCs. Further, note that a step can be associated with multiple KCs in which case, we can just unroll the step to ensure that each step has a single KC. While it is possible to adapt our approach to perform structure prediction where instead of a single KC, a step can be mapped to a more complex structure (e.g. a graph), we leave this for future work and focus on the case where a single KC is mapped to a step in the strategy.

In this paper, the task that we want to solve is the following. Given a student s and a problem p, we predict the sequence of KCs that s will use to solve p. In particular, we assume that we have a large dataset $\mathcal{D}$ where we refer to an *instance* in the dataset as a pair $(s, p) \in \mathcal{D}$. We want to sample instances from $\mathcal{D}$ to train a model that takes as input $(s, p) \in \mathcal{D}$ and predicts strategies, i.e., variable-length sequences of KCs. We also assume that $\mathcal{D}$ contains correctness associated with each step in the strategies. Specifically, for an input (s, p), for each step that s takes to solve p, we know if s was successful in solving that step correctly. We use this information to determine the mastery of a student and based on this, we develop an embedding (vector representation) for students and problems. We then jointly cluster the embeddings using a non-parametric approach such that instances where the strategies are approximately symmetric are clustered together. Finally, we train an LSTM model to predict strategies by sampling the clusters. In the subsequent subsections, we first describe our embedding called MVec. Next, we apply DP-Means HDP clustering [10] to the embeddings while also incorporating approximate symmetries in strategies.

3.1 MVec Embeddings

To learn the MVec embedding, we use an approach that is similar to Node2Vec [5]. Specifically, we construct a relational graph $\mathcal{G} = (\mathbf{V},\mathbf{E})$ as follows. Each student, problem, and KC in the training data is represented as a node $V \in \mathbf{V}$. For every student S who uses KC K as a step to solve problem P, there exist 2 edges $E, E' \in \mathbf{E}$, where E connects the node representing the student to the node representing the KC and E' connects the node representing the KC to the node representing the problem. An example graph over 3 students, problems, and KCs is shown in Fig. 2. We now sample paths in the graph and learn embeddings for

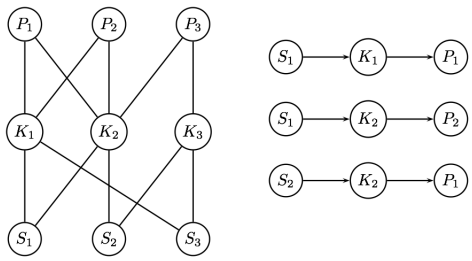

Figure 2: Illustrating a graph network of three students, problems, and KCs. The figure on the right shows some of the sampled random walks/paths.

these paths using word embedding models (Word2Vec) [14]. Specifically, the objective function is as follows.

$$\max_{f} \sum_{u \in \mathbf{V}} logP(N_Q(u)|f(u)) \tag{2}$$

where $f : u \rightarrow \mathbb{R}^d$ is the vector representation for nodes $u \in \mathbf{V}$, $N_Q(u)$ denotes the neighbors of u sampled from a distribution Q. Similar to Node2Vec, we assume that there is a factorized model that gives us a conditional likelihood that is identical to the likelihood function used in Word2Vec.

$$P(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in \mathbf{V}} \exp(f(v) \cdot f(u))} \tag{3}$$

where n_i is a neighbor of u. The conditional likelihood is optimized by predicting neighbors of u using u as input in an autoencoder neural network. The hidden layer learns similar embeddings for nodes with symmetrical neighborhoods. To do this, we generate walks on $\mathcal{G}$ as shown for the example in Fig. 2, and in each walk, given a node, we predict neighboring nodes similar to predicting neighboring words in sentences. To generate these walks, a simple sampling strategy Q is to randomly sample a neighbor for a node. However, in our case, it turns out that each neighbor may have different importance when it comes to determining symmetry. Specifically, if a student has achieved mastery in applying a KC to a problem, then the corresponding edges should be given greater importance when determining symmetry between nodes in $\mathcal{G}$. To do this, we train a Sequence-to-Sequence attention model [31] from which we estimate the sampling probabilities for edges in $\mathcal{G}$.

The intuitive idea in quantifying mastery is illustrated in Fig. 3 which shows the opportunities given to 3 students to apply KCs in different problems. For each sequence of KCs, we predict if the student got the step correct or wrong on the first attempt (abbreviated as CFA for Correct First Attempt) when given an opportunity to apply the KC. The CFA values are performance indicators for the student, i.e., if they have mastered a KC, then they are likely to get the step correct in every opportunity they get to apply that KC. We train a model to predict the CFA values (CFA = 1 indicates a correct application of the KC) given the KCs used in a problem. The predicted values from the model are shown for each KC. The bar graphs show mastery over the KCs. As seen here, the first student is inconsistent in applying

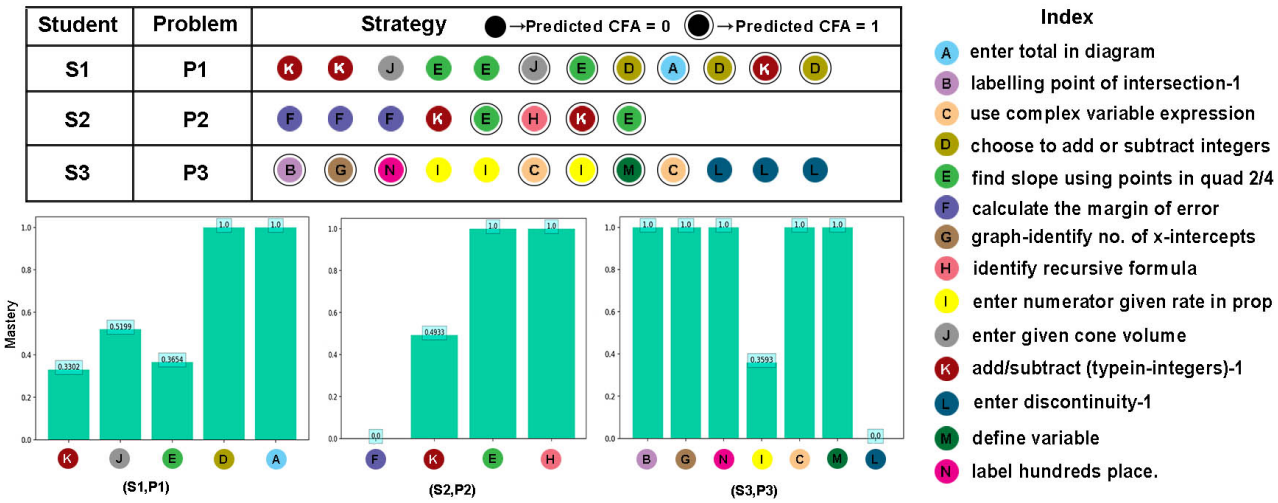

Figure 3: An example to illustrate the use of attention for mastery estimation. The bar charts show for each KC, the attention on a KC across steps that the student solves successfully (CFA=1) normalized by total attention for that KC. Larger values indicate that the model believes the student understands the KC as the attention on it is large when CFA=1 and vice versa.

the skill, *find the slope using points* (labeled as E) since the predictions for this oscillate between 0 and 1 whenever the student tries to apply this KC. On the other hand, student 2 consistently applies the same skill correctly and therefore the attention value is higher. We train the attention model from opportunities based on curriculum structure. Specifically, the curriculum consists of multiple units and each unit is further subdivided into sections. For each student S, from every unit that the student has completed say U, we select a problem P from each section that the student has worked on in U and train the model to predict the CFA values for each KC used in P. We use the standard architecture described in [31] for this model. Specifically, the input consists of the KC sequence, and the encoder maps this sequence to a latent representation and the decoder decodes the CFA values one at a time. The attention is given by

$$Attention(\gamma, \kappa, \eta) = softmax\left(\frac{\gamma \kappa^T}{\sqrt{d_k}}\right)\eta \tag{4}$$

where γ, κ, and η are the standard query, key, and value matrices respectively as defined in [31], and d_k is the dimensionality of the embedding that represents the latent representations. We use the encoder-decoder attention, i.e., the query is the decoder representation and the key is the encoder representation. The attention weights are an estimate of the alignment between encoded latent representations of mastery with the decoded representation of correctly applying a skill at each step in the problem. The projection of mastery over a KC K based on the attention vectors is estimated by the following equation.

$$\alpha(S, P, K) = \frac{\sum_i \sum_{v \in \pi(a_i)} v}{\sum_i \sum_{v \in \pi(a_i)} v + \sum_i \sum_{v' \in \bar{\pi}(a_i)} v'} \tag{5}$$

where $\pi(\cdot)$ extracts only those values in the input vector where the corresponding output for that step is predicted as 1, i.e., the model predicted that the student could solve the step correctly. $\bar{\pi}(\cdot)$ is the complement of $\pi(\cdot)$, i.e., it extracts attention values corresponding to steps that were predicted as mistakes made by the student and i sums up all the instances where K is used.

We now sample paths from $\mathcal{G}$ using the factored distribution, i.e., $Q(S) * Q(K|S) * Q(P|K,S)$, where $Q(S)$ is the probability of sampling a student node, $Q(K|S)$ is the probability of sampling a KC K given student S and $Q(P|K,S)$ is the probability of sampling problem P given K, S. We assume that $Q(S)$ is a uniform distribution over students. The conditional distributions are as follows.

$$Q(K|S) = 1/n \sum_p \alpha(S, P, K) \tag{6}$$

$$Q(P|K,S) = \alpha(S, P, K) \tag{7}$$

where n is the number of opportunities given to student S to apply KC K. The algorithm to generate MVec embeddings is shown in Algorithm 1. As shown here, we sample a path in the graph as follows. We first sample student S uniformly at random, then we sample a KC K from $Q(K|S)$ and a problem from $Q(P|K,S)$. We then predict each node in the path using the neighboring nodes through a standard Word2Vec model. The resulting embeddings are learned in the hidden layer of the Word2Vec model. Note that for scalability, we do not construct/store the full graph $\mathcal{G}$ at any point. Instead, we only sample paths in an online manner as shown in Algorithm 1.

3.2 Non-Parametric Clustering

We cluster the student and problem MVec embeddings jointly through a non-parametric approach based on symmetries defined as follows. For the dataset denoted by $\mathcal{D}$, let $\mathbf{S}$, $\mathbf{P}$ denote the set of students and problems respectively in $\mathcal{D}$.

Definition 1. *A* strategy-invariant *partitioning w.r.t $\mathcal{D}$ is a partitioning $\{\mathbf{S}_i\}_{i=1}^{k_1}$ and $\{\mathbf{P}_j\}_{j=1}^{k_2}$ such that $\forall i, j$, if $S, S' \in \mathbf{S}_i$, $P, P' \in \mathbf{P}_j$, S, S' follow equivalent strategies for P, P' respectively.*

where k_1 and k_2 are the number of partitions/clusters for

Algorithm 1 Generate MVec embeddings

Input: Relation Graph: $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ with student, problem and KCs as nodes, Embedding dimension: d, pre-trained attention-model $\mathcal{A}$
Output: Embeddings for each node $v \in \mathbb{R}^d$
Initialize: set of walks, $\mathcal{W} = empty$
1. **for all** $t = 1$ to T **do**
2. Sample a path $< S, K, P >$ in $\mathcal{G}$ from $Q(S) * Q(K|S) * Q(P|K, S)$ using Eq. (6) and (7).
3. $\mathcal{W} = \mathcal{W} \cup < S, K, P >$
4. **end for**
5. $v_e = word2vec(\mathcal{W}, d)$
6. **return** v_e

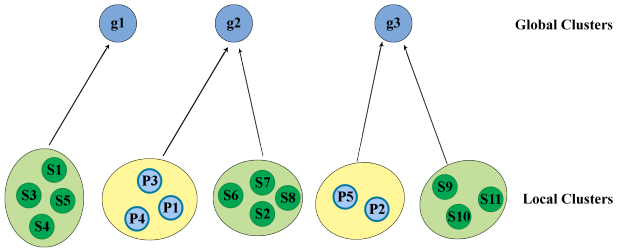

Figure 4: HDP Clustering showing the local clusters (student clusters and problem clusters) and the global clusters that combine the student, problem clusters.

students and problems respectively. The benefit of strategy-invariant partitioning is that we can scale up without sacrificing accuracy by training a prediction model only on samples drawn from the partitions instead of the full training data. Therefore, our task is to learn such partitioning approximately (since constraining the partitions to have exact equivalence of strategies is a hard problem). Since it is hard to know apriori how many partitions are needed, we formulate this as a non-parametric clustering problem and use DP-Means [10] to learn the clusters.

To formalize our approach, we begin with some notation. Let $\mathbf{S} = \{x_{i1}\}_{i=1}^{N}$ denote the set of students and $\mathbf{P} = \{x_{j2}\}_{j=1}^{M}$ denote the set of problems. We refer to the student and problem clusters as the *local* clusters. A *global* cluster combines student and problem clusters as illustrated in Fig. 4. We run the standard DP-Means HDP clustering algorithm to optimize Eq. (1) and learn global clusters that combine local clusters over $\mathbf{S}$ and $\mathbf{P}$. Note that large values of the global penalty λ_g result in a coarse clustering with few clusters and small values of the penalty result in fine-grained clusters. We adaptively change λ_g where we progressively lower the penalty yielding a *coarse-to-fine* refinement of the clusters. Specifically, suppose $\ell_1 \ldots \ell_g$ are the current global clusters, we compute a score $\mathcal{S}(\ell_1 \ldots \ell_g)$ based on the symmetry of strategies within each cluster and as long as the score progressively improves across iterations, we reduce λ_g to obtain finer-grained clusters.

3.3 Refining Clusters using Symmetry

Note that each global cluster implicitly represents a set of strategies, i.e., a student-problem pair (s, p) within the cluster corresponds to a strategy followed by s for problem p.

Algorithm 2 Coarse-to-Fine Refinement

Input: Student/Problem set: $\{x_{ij}\}$, Constant Penalty parameter: λ_ℓ, iteration limit T
Output: Global strategy clustering $\{\ell_1, \ldots, \ell_g\}$
Initialize : Global cluster penalty $\lambda_g = y$ (where y is a large number), $t = 0$, cluster coherence $coh_{t-1} = 0$.
1. **repeat**
2. $t = t + 1$
3. Cluster with penalties λ_ℓ, λ_g
4. Compute cluster coherence score, $coh_t = \mathcal{S}(\ell_1, \ldots \ell_g)$
5. Reduce: $\lambda_g = \lambda_g - \epsilon$
6. **until** $coh_t > coh_{t-1}$ or $t > T$

We want to quantify symmetry between strategies within a cluster. A naive approach to compare two strategies is to compute the mean of the MVec embeddings for the KCs used in each strategy and then compute the distance between the means. However, this assumes that all permutations of a strategy are equivalent to each other which is problematic. On the other hand, suppose we compare the KC embedding at a step in one strategy with the KC embedding at the same step in the other strategy, then we assume the strategies are equivalent only if they are perfectly aligned with each other which is also an over-simplification.

To match strategies approximately, we represent a strategy using a combination of embeddings and positional encodings [31], and approximately align two strategies to estimate the symmetry between them. A KC K in the strategy is represented by its positional embedding $\vec{K} = \vec{K_e} + \vec{K_p}$ where $\vec{K_e}$ is the MVec embedding for K and $\vec{K_p}$ is the positional encoding for K in the strategy. To compute symmetry between strategies, we compute an alignment between their positional embeddings. Alignment is a fundamental problem in domains such as Bioinformatics where a classical approach that is often used is the Smith and Waterman algorithm (SW) [28]. The idea is to perform local search to compute the best possible alignment between two sequences. SW requires a *similarity function* which in our case is the similarity between two KCs and we set this to be $s(K, K')$ $= \vec{K}^\top \vec{K'}$, i.e., the cosine similarity between the positional embeddings of K and K'. Further, SW also requires a *gap penalty* which refers to the cost of leaving a gap in the alignment. In our case, we set the gap penalty to 0 since we want symmetry between strategies to be invariant to gaps. That is, if two strategies are symmetric, adding extra steps in the strategies is acceptable. SW iteratively computes a scoring matrix based on local alignments. The worst-case complexity to compute the scoring matrix that gives us scores for the best alignment is equal to $O(m * n)$ where m and n are lengths of the strategies.

Note that in our case, we are interested in quantifying symmetry between strategies based on the alignment. Specifically, let $\mathbf{K}$ and $\mathbf{K'}$ be two strategies of lengths n and m respectively. SW gives us an alignment between $\mathbf{K}$ and $\mathbf{K'}$ denoted by $L(\mathbf{K}, \mathbf{K'})$. The alignment consists of the pairs KCs from $\mathbf{K}$ and $\mathbf{K'}$ respectively that have been matched/aligned or a gap, i.e., a KC from $\mathbf{K}$ could not be aligned with any KC from $\mathbf{K'}$. We compute the symmetry score between $\mathbf{K}$ and $\mathbf{K'}$ as $r(\mathbf{K}, \mathbf{K'}) = \frac{1}{\max(n,m)} \sum_{(K,K') \in L(\mathbf{K},\mathbf{K'})} (\vec{K}^\top \vec{K'})$, where

$(K, K') \in L(\mathbf{K}, \mathbf{K}')$ are aligned KCs and $\vec{K}^\top \vec{K}'$ is their cosine similarity. We see that $0 \leq r(\mathbf{K}, \mathbf{K}') \leq 1$. Based on this, we estimate symmetry in the clustering as follows.

$$\mathcal{S}(\ell_1, \ldots, \ell_g) = \frac{1}{g} \sum_{p=1}^{g} \mathbb{Z}_p \sum_{\mathbf{K}, \mathbf{K}' \in T(\ell_p)} r(\mathbf{K}, \mathbf{K}') \qquad (8)$$

$T(\ell_p)$ is a set of all strategies in ℓ_p and $\mathbb{Z}_p = \frac{2}{|T(\ell_p)|(|T(\ell_p)|-1)}$ is the normalization term. Thus, a larger value of $\mathcal{S}(\ell_1, \ldots, \ell_g)$ implies that the clustering corresponding to $\ell_1, \ldots, \ell_g$ has a greater degree of symmetry in strategies. Using this score, we refine the clustering by adapting the global penalty. Specifically, we reduce the global penalty λ_g by a constant ϵ as long as the symmetry score decreases across iterations or for a fixed number of iterations. Algorithm 2 summarizes the coarse-to-fine refinement.

3.4 Training the Model

We use an LSTM architecture similar to [27] to predict strategies. Specifically, the model is a one-to-many LSTM that takes student, problem vectors as input and generates a sequence of KCs as output. To train this model, we sample instances from the converged global clusters, and for each sampled student-problem pair (s, p), the LSTM input is the concatenation of MVec embeddings of s and p. The output corresponds to the sequence of KCs in the strategy used by s for p, each of which is encoded as a one-hot vector. To handle variable-length strategies, a special *stop* symbol is used to denote the end of a sequence. The entire model is trained using the standard categorical cross-entropy loss.

4. EXPERIMENTS

Our goal is to answer the following questions through our evaluation. i) what is the accuracy of our approach in predicting strategies? ii) how does our approach scale-up? iii) what is the influence of mastery in predicting strategies accurately? and iv) is there a disparity in the accuracy of prediction for different skill-based sub-groups in the data?

4.1 Dataset

The data we use in this work is large-scale real-world education data recorded with real students using MATHia. MATHia is an online math learning program for middle school students that is popularly used across several schools. We used two datasets provided by MATHia for evaluating our proposed approach, Bridge-to-Algebra 2008-09 (`BA08`) and Carnegie Learning MATHia 2019-20 (`CL19`). Both of these datasets contain recorded interactions between the student and the computer tutor while the student attempts to solve a problem on the platform. Each recorded interaction consists of the log of the student's action toward solving the problem, for example, the knowledge component used, if hints were needed and if the step was completed correctly. `BA08` is an older dataset that consists about 20 million interactions for about 6000 students and $52k$ unique algebra problems. This dataset contains about 1.6 million *data instances*. It is important to note that we consider a *data instance* as a student-problem pair, so one *data instance* consolidates all the interactions/steps for one student on a specific problem. `CL19` is a more recent and larger dataset containing about 47 million interactions for 5000 students and about $32k$ unique math problems. It has about 1.9 million *data instances*. Both datasets are publicly available through the PSLC datashop [30].

Table 1: Main parameters for the models.

Transformer-based model	LSTM-based strategy model
Dimension → 512	Latent Dimension → 200
Number of layers → 6	Epochs → 60
Number of heads → 8	Batch Size → 30
Dimensions of key, value and query → 64	Adam Optimizer with Learning rate 0.01
Max Sequence Length → 150	Dropout → 0.1
Dropout → 0.1	
Weight Sharing → False	

4.2 Experimental Setup

To train the attention model, we used the transformer implementation in [31]. For the strategy prediction, we used a one-to-many LSTM [27] where the input is the student and problem embedding, and the output is the sequence of KCs. The parameters for the two models are shown in Table 1. We used the standard parameters for the transformer model and retained the same parameters as in [27] for the LSTM model for an unbiased comparison. For generating MVec embeddings, we used Gensim [14] with an embedding dimension set to 300 (which is typically used). We initialize the local cluster penalty $\lambda_\ell = 7$ and global cluster penalty $\lambda_g = 9$ for Coarse-to-Fine refinement and reduce the global penalty by $\epsilon = 1$ (we discovered these to be the best-performing hyper-parameters in experimentation). We perform our experiments on a machine with 64 GB RAM, an Nvidia Quadro 5000 GPU with 16 GB memory, and a CPU with 8 cores. The code for our implementation is available here [1].

4.3 Comparison to Baselines

We compared our approach with the following methods. The first one is a specialized approach proposed in Shakya et. al. [27] (CS) for the same datasets where an LSTM is trained using importance sampling. However, this sampling does not incorporate mastery or approximate symmetries to find diverse training instances. We also applied a more general importance sampling approach that is said to be applicable for any DNN model training proposed in [7] (IS) using their publicly available implementation. However, IS failed to output any results for datasets of our size and therefore we do not show it in our result graphs. This indicates that general-purpose methods do not scale up for our datasets. We also developed a stratified sampler (GS for group sampling) where the distribution is only proportional to the number of problems solved by a student, i.e., we sample more instances from students that have data associated with them. The last baseline is a naive Random Sampler (RS) used as a validation check where we sample students and problems uniformly at random. We refer to our approach as Attention Sampling (AS). In our evaluation, for each approach, we enforce a limit on the number of training instances and measure test accuracy based on the model trained with this limit. This is similar to a measure of the *effective model complexity* [17] which is the number of training samples to achieve close to zero error. We report the average accuracy of predicted KCs based on three training runs.

[1] `https://github.com/anupshakya07/attn-scaling`

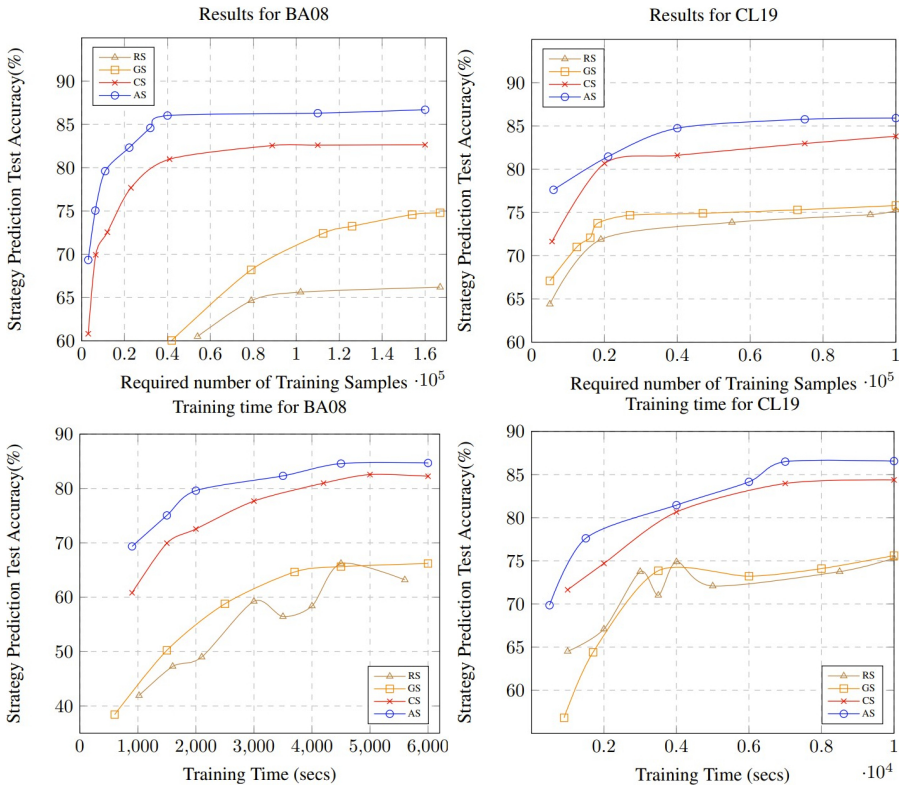

Figure 5: Illustrating Scalability vs Accuracy. (a), (b) show test accuracy for strategy prediction for varying training datasize limits. (c), (d) show accuracy (strategy prediction) for different training time limits.

4.4 Results and Discussion

4.4.1 Accuracy

The strategy prediction accuracy results for `BA08` and `CL19` are shown in Fig. 5 (a) and (b). As shown in Fig. 5 (a), for `BA08`, in our approach (AS), it takes less than 1% of the entire data (of `BA08` containing 1.6 million instances) to obtain test accuracy that is greater than 80%. CS is the next best performer but is consistently below AS for all training sizes. GS performs significantly worse which illustrates that symmetries are more complex and a simple grouping based on problems/students is insufficient. The poor performance of RS validates that the problem of choosing the correct samples is a challenging one. As seen in Fig. 5 (b), for a considerably larger dataset `CL19`, we can observe similar performance as in `BA08`. AS remains the best performer and here CS is less stable since we see a performance drop as we increase the limit on training samples. This suggests that CS may not be able to capture all symmetries and thus may produce a more biased training sample set. The results for GS and RS are similar to those observed in `BA08`. As mentioned before, IS failed to produce any results.

4.4.2 Scalability

Fig. 5 (c) and (d) show the training time required to obtain a specific accuracy for `BA08` and `CL19` respectively. Even with the extra processing that is needed to compute the mastery-based embeddings and the non-parametric clustering, AS requires the shortest training time to achieve an accuracy that is higher than the other approaches. This illustrates the significance of leveraging symmetries in the data to train the model. As mentioned before, the full data is infeasible to train and when attempting to use the full data, the model did not converge for both datasets even after several days of training time using our experimental setup. As seen in our results, for `CL19`, the training time is larger since it takes longer to compute the groups using non-parametric clustering due to the much larger size of the dataset. However, considering that `CL19` is significantly larger than `BA08`, we see that AS could still scale up to this dataset quite easily while IS which is a state-of-the-art sampling method for DNN training failed to train the model.

Table 2: Ablation study with NS (No symmetries used), SS (Symmetries without using mastery) and MS (Adding the mastery model to better identify symmetries). Results are shown for 2 datasets with different sample sizes. Accuracy results in %.

Expts.	BA08			CL19		
	40k	100k	150k	40k	80k	100k
NS	60.05	71.14	74.58	74.81	75.4	75.8
SS	80.98	82.3	82.65	81.6	83.2	83.8
SS + MS	86.02	86.21	86.53	84.74	85.8	85.9

4.4.3 Ablation Study

Table 2 shows the results of our ablation study. We add each component to our overall approach and observe the test accuracy as we vary the sample size in the training data. Specifically, the first case (**NS**) uses no symmetries,

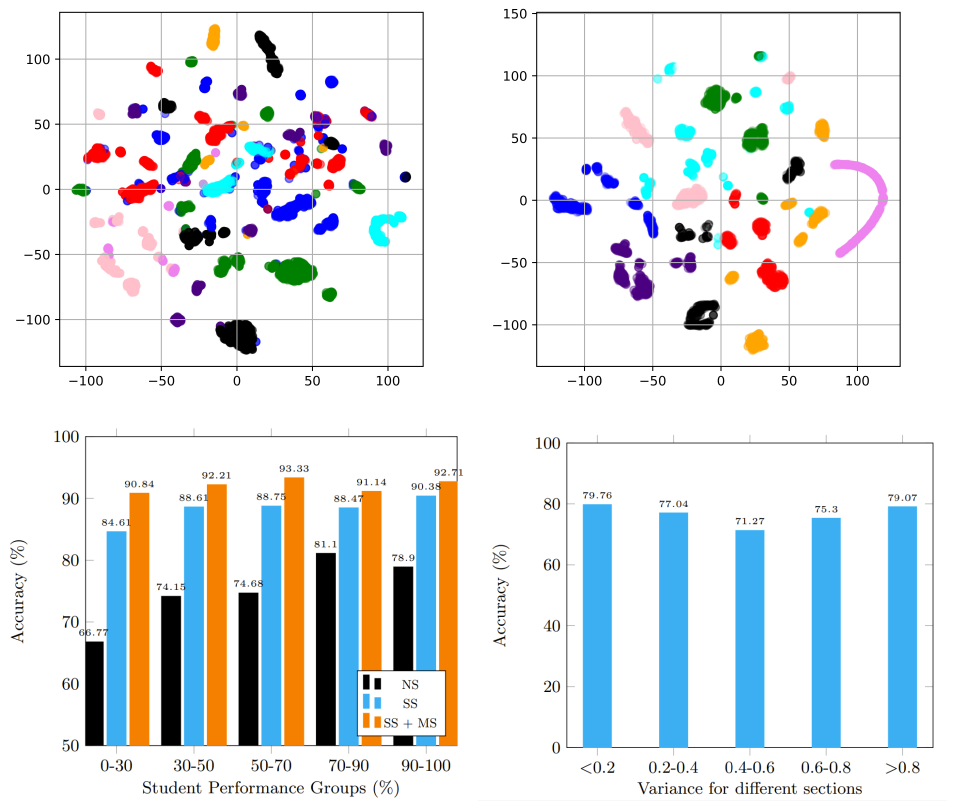

Figure 6: T-SNE visualization of strategy clusters for `CL19`. The color-coded plots show the 2D representation of the different strategy clusters for (a) Embeddings that do not use mastery (b) MVec embeddings. The strategy representations are extracted from the final hidden layer of the LSTM model and converted to 2D representation using T-SNE. (c) shows accuracy for different groups of students (based on their performance) for `CL19`. The x-axis denotes different ranges of %s, where a range $a - b$ denotes that students in this group got $> a$ and $< b$ steps correct in their first attempt. The y-axis shows accuracy over the groups. (d) shows the performance of the model on different groups based on the average variance of the strategies in the sections for `CL19`. Variance is computed using edit distance as the metric of similarity between strategies.

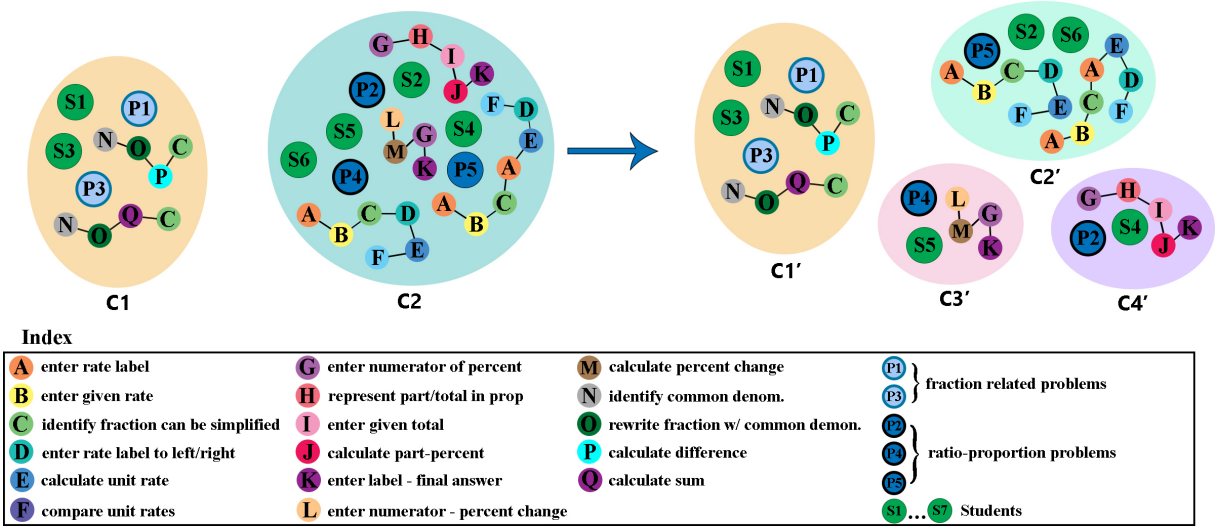

Figure 7: An example from the dataset CL19 illustrating coarse-to-fine refinement of clusters. Strategies are shown by paths connecting KCs. **C1** and **C2** are the coarse clusters which get refined into strategy invariant clusters **C1′**, **C2′**, **C3′** and **C4′**.

Table 3: Different strategies used by the students for different problems in the same section for `CL19` dataset. The model is able to predict accurately as student adapt their strategies.

Student	Problem Name	Predicted Strategy	Actual Strategy
S_1	linear inequalities numberline 5	represent open point on numberline-1 represent ray on numberline-1 represent inequality in symbolic problem-1 identify when finished with numberline-1 identify invisible non-inflection point is in solutionset-1 identify invisible non-inflection point is not in solutionset-1 identify visible non-inflection point is not in solutionset-1	represent open point on numberline-1 represent ray on numberline-1 represent inequality in symbolic problem-1 identify when finished with numberline-1 identify invisible non-inflection point is in solutionset-1 identify invisible non-inflection point is not in solutionset-1 identify visible non-inflection point is not in solutionset-1
	linear inequalities numberline 9	write simple inequality in verbal problem-1 represent closedpoint on numberline-1 represent ray on numberline-1 identify when finished with numberline-1 identify visible non-inflection point is not in solution set-1 identify invisible non-inflection point is not in solution set-1 identify inflection point in solution set-1	write simple inequality in verbal problem-1 represent closedpoint on numberline-1 represent ray on numberline-1 identify when finished with numberline-1 identify visible non-inflection point is not in solution set-1 identify invisible non-inflection point is not in solution set-1 identify inflection point in solution set-1 identify invisible non-inflection point is not in solution set-1 identify inflection point in solution set-1
S_2	ratio proportion prop1 4	enter part in proportion with variable-1 enter given total in proportion-1 enter numerator of given rate in proportion-1 enter denominator of given rate in proportion-1 enter proportion label in numerator-1 enter proportion label in denominator-1 calculate part in proportion with fractions-1 enter numerator of form of 1-1 enter denominator of form of 1-1 enter calculated value of rate-1	enter part in proportion with variable-1 enter given total in proportion-1 enter denominator of given rate in proportion-1 enter numerator of given rate in proportion-1 enter proportion label in numerator-1 enter proportion label in denominator-1 calculate part in proportion with fractions-1 enter denominator of form of 1-1 enter numerator of form of 1-1 enter calculated value of rate-1
	ratio proportion prop1 5	enter proportion label in numerator-1 enter proportion label in denominator-1 enter given total in proportion-1 enter numerator of given unit rate in proportion-1 enter denominator of given unit rate in proportion-1 calculate part in proportion with fractions-1 enter calculated value of rate-1	enter proportion label in numerator-1 enter proportion label in denominator-1 enter given total in proportion-1 enter numerator of given unit rate in proportion-1 enter denominator of given unit rate in proportion-1 calculate part in proportion with fractions-1 enter calculated value of rate-1

i.e., the clustering is performed randomly. Next, we cluster based on embeddings without using the mastery, i.e., when we generate the embeddings for MVec, we do not use the attention model and simply use triplets (S, P, K), where S is a student, P is a problem and K is a KC used by S for P as input to Word2Vec and generate embeddings. Thus, we use symmetries in strategy without utilizing mastery when we generate the clusters. We show this as Strategy Symmetry (**SS**) in the table. Finally, we add mastery to generate embeddings denoted by **SS** + **MS** and as shown, this improves the generalization performance for all sample-sizes thus, illustrating that utilizing mastery to learn embeddings plays a significant role in improving accuracy in predicting strategies.

4.4.4 *Visualizing Clusters*

We used T-SNE to visualize the clusters of strategies. For this, we pick 100 student-problem pairs sampled from 10 clusters. We then perform strategy prediction for these and visualize the hidden-layer representation of the LSTM in the T-SNE plot. We compare this for MVec embeddings as well as embeddings that are learned without using mastery. As shown in Fig. 6 (a) and (b), when we use MVec, the LSTM hidden-layer representation of strategies has better separation. This indicates that we learn better grouping of strategies using MVec embeddings.

4.4.5 *Fairness*

We evaluate if our approach results in disparate mistreatment. Specifically, this means that the model should not have significantly different accuracy for different sensitive sub-groups in the data. In our case, the sensitive sub-groups correspond to students at different skill levels. That is, we want to predict the strategies equally well for all students. To do this, we conducted an experiment where we divide the test data into 6 performance groups. The performance groups are based on the % of problem steps the students solve correctly on their first attempt. The performance groups include students who scored in the following ranges $\leq 30\%$, $30-50\%$, $50-70\%$, $70-90\%$, $\geq 90\%$. To measure disparate mistreatment, we compare the average accuracy of strategies predicted for each of these groups. For a student S in performance group $\mathbf{G}$, we predict the strategies for all problems attempted by S in the test set and measure the average accuracy μ_S. We then compute the accuracy over a performance group as $1/|\mathbf{G}| \sum_{S \in \mathbf{G}} \mu_S$. Fig. 6(c) shows our results for the variants, NS, SS and SS+MS (identical to those used in the ablation study) for CL19 (we show results on this since this is the larger and more recent dataset). As seen from our results, SS+MS yields the best accuracy over each performance group. Further, the accuracy over the poorest and the best performers is comparable to each other and not significantly different. Thus, there is no disparate mistreatment of any performance group shown by our approach.

Next, we want to verify if there is disparate mistreatment when we consider sub-groups that have rare strategies. To measure this, we divided the problem sections in the test set into groups based on the variance among strategies for problems in those sections. Specifically, to perform worst-case analysis, we used the edit distance to measure the variance of strategies within problems in a section. That is, if a pair

of problems vary in two out of 10 steps, the edit distance is 0.2. We computed the variance in edit distances over all the problems in a section. We then obtained the sub-groups at 5 different thresholds of variance. Thus, groups that have large variance include more rare strategies, while groups that have smaller variance have fewer rare strategies. For all the problems in each of these sub-groups, we computed the average accuracy in strategy prediction. Fig.6(d) shows our accuracy results over all the sub-groups. As seen here, we have no disparate mistreatment for any of the sub-groups. Thus, we show that even in cases where rare strategies are used by students, our approach predicts strategies with an accuracy that is very similar to cases where common strategies are used.

4.4.6 Example Cases

Table 3 illustrates examples corresponding to two different students where we predict strategies for two problems taken from the same section in each case. Note that the students make modifications to their strategy to suit the problem context as seen in the examples, though the overall strategies are similar since the problems are from the same section. The model is able to successfully adapt and predict these strategy changes quite accurately. In the case of student S_1, for the second problem, the model predicted most of the steps except that the student had some redundant steps at the end which were not predicted by the model. In the case of S_2, for problem 1, the predicted strategy interchanged the order of a couple of steps that clearly does not significantly alter the underlying strategy.

We illustrate some examples of coarse-to-fine refinement in Fig. 7. Specifically, we show examples from two types of problems, *Fractions* and *Ratio, Proportions*. The clusters indicate the students, problems, and strategies followed by students. In cluster **C1**, even though there are two different strategies, they are symmetric to each other and therefore, in a subsequent iteration of refinement, **C1′** is the same as **C1**. On the other hand, **C2** consists of 4 strategies, 2 of these are expert-level strategies and the other two are simpler but differing strategies. Upon refinement of **C2**, we get **C2′** which intuitively represents the expert students and **C3′**, **C4′** which represents students using simpler yet different strategies. Thus, the coarse-to-fine refinement results in invariant strategies within each cluster.

5. CONCLUSION

We presented a scalable and equitable framework for predicting math problem-solving strategies used by students. Since students with differing skill levels use significantly different strategies, to predict these, we need to train a model over diverse training instances. Identifying such instances is a challenging problem in big data. Particularly, identifying strategies which are approximately symmetrical to each other is a hard task. Here, we developed a clustering approach to discover diverse groups where instances within each group have approximately symmetrical strategies. Specifically, we learned an embedding MVec using a combination of Node2Vec where we learned representations for relationships in the data encoded as a graph and a transformer model that predicts mastery. Specifically, similar attentions in the transformer model over steps in the strategy indicated similar mastery in solving a problem, which we used to learn the Node2Vec representation. We then clustered the MVec embeddings with a non-parametric algorithm called DP-Means by iteratively refining the clusters based on the level of symmetry encoded within the clusters. By sampling from clusters, we were able to train an LSTM model to predict strategies using small but highly informative instances that were representative of strategies in the full data. Further, by sampling from clusters, we ensured that the LSTM model did not optimize its parameters for any specific group, but instead generalized over all groups in the data, thus making the model capable of identifying strategies from diverse groups. Experiments on two large-scale datasets demonstrated our accuracy in predicting strategies with a small fraction of the dataset and further, our predictions were fair across students at different levels of skill.

As part of future work, we hope to extend this model to non-structured interactions (e.g. conversations). Further, we also plan to explore more complex mappings of strategies where each step can be represented by a structure (e.g. graph, table, etc.) and developing structured prediction models from such mappings. We also propose to utilize this approach in instructional design where we can select problems to solve based on a student's predicted strategy and also to develop interventions in ITSs based on misconceptions identified in predicted strategies.

6. ACKNOWLEDGEMENT

This research was sponsored by the National Science Foundation under NSF IIS award #2008812 and NSF award #1934745. The opinions, findings, and results are solely the authors' and do not reflect those of the funding agencies.

7. REFERENCES

[1] B. S. Bloom. Learning for mastery. instruction and curriculum. regional education laboratory for the carolina and virginia, topical papers and reprints, number 1. *Evaluation comment*, 1(2):n2, 1968.

[2] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021.

[3] Y. Chen, P.-H. Wuillemin, and J.-m. Labat. Discovering prerequisite structure of skills through probabilistic association rules mining. In *Proceedings of the 8th International Conference on Educational Data Mining*, pages 117–124, 06 2015.

[4] A. T. Corbett. Cognitive computer tutors: Solving the two-sigma problem. In *Proceedings of the 8th International Conference on User Modeling 2001*, page 137–147, 2001.

[5] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 855–864, 2016.

[6] A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2525–2534, 2018.

[7] A. Katharopoulos and F. Fleuret. Processing

megapixel images with deep attention-sampling models. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3282–3291. PMLR, 2019.

[8] K. R. Koedinger, A. T. Corbett, and C. Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cogn. Sci.*, 36:757–798, 2012.

[9] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. *ICML*, 2012.

[10] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[11] K. Leelawong and G. Biswas. Designing learning by teaching agents: The betty's brain system. *Int. J. Artif. Intell. Ed.*, 18(3):181–208, Aug. 2008.

[12] Y. Liu, Y. Yang, X. Chen, J. Shen, H. Zhang, and Y. Yu. Improving knowledge tracing via pre-training question embeddings. *arXiv preprint arXiv:2012.05031*, 2020.

[13] N. Maharjan, D. Gautam, and V. Rus. Assessing free student answers in tutorial dialogues using LSTM models. In *Artificial Intelligence in Education - 19th International Conference, AIED*, pages 193–198, 2018.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Systems*, pages 3111–3119, 2013.

[15] D. M. Morrison, B. Nye, V. Rus, S. Snyder, J. Boller, and K. B. Miller. Tutorial dialogue modes in a large corpus of online tutoring transcripts. In *Artificial Intelligence in Education - 17th International Conference*, volume 9112, pages 722–725. Springer, 2015.

[16] H. Nakagawa, Y. Iwasawa, and Y. Matsuo. Graph-based knowledge tracing: modeling student proficiency using graph neural network. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 156–163, 2019.

[17] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. Deep double descent: Where bigger models and more data hurt. In *ICLR*, 2020.

[18] S. Pandey and G. Karypis. A self attentive model for knowledge tracing. In *EDM*. International Educational Data Mining Society (IEDMS), 2019.

[19] B. E. Penteado. Estimation of prerequisite skills model from large scale assessment data using semantic data mining. In *International Conference on Educational Data Mining*, pages 675–677, 2016.

[20] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, and J. Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[21] S. Ritter. Communication, cooperation and competition among multiple tutor agents. In *Artificial Intelligence in Education: Knowledge and media in learning systems*, pages 31–38, 1997.

[22] S. Ritter, R. Baker, V. Rus, and G. Biswas. Identifying strategies in student problem solving. *Design Recommendations for Intelligent Tutoring Systems*, 7:59–70, 2019.

[23] S. Ritter, M. Yudelson, S. E. Fancsali, and S. R. Berman. How mastery learning works at scale. In *L@S*, pages 71–79. ACM, 2016.

[24] Y. Roh, K. Lee, S. E. Whang, and C. Suh. Fairbatch: Batch selection for model fairness. In *9th International Conference on Learning Representations, ICLR*, 2021.

[25] V. Rus, S. K. D'Mello, X. Hu, and A. C. Graesser. Recent advances in conversational intelligent tutoring systems. *AI Magazine*, 34(3):42–54, 2013.

[26] V. Rus, N. Maharjan, L. J. Tamang, M. Yudelson, S. R. Berman, S. E. Fancsali, and S. Ritter. An analysis of human tutors' actions in tutorial dialogues. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, pages 122–127, 2017.

[27] A. Shakya, V. Rus, and D. Venugopal. Student strategy prediction using a neuro-symbolic approach. In *Proceedings of the 14th International Educational Data Mining Conference (EDM 21)*, 2021.

[28] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[29] X. Song, J. Li, Q. Lei, W. Zhao, Y. Chen, and A. Mian. Bi-clkt: Bi-graph contrastive learning based knowledge tracing. *Knowledge-Based Systems*, 241:108274, 2022.

[30] J. C. Stamper, K. R. Koedinger, R. S. J. de Baker, A. Skogsholm, B. Leber, S. Demi, S. Yu, and D. Spencer. Datashop: A data repository and analysis service for the learning science community. In *AIED*, volume 6738, page 628, 2011.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[32] D. Venugopal and V. Rus. Joint inference for mode identification in tutorial dialogues. In *COLING 2016, 26th International Conference on Computational Linguistics*, pages 2000–2011. ACL, 2016.

[33] D. Venugopal, V. Rus, and A. Shakya. Neuro-symbolic models: A scalable, explainable framework for strategy discovery from big edu-data. In T. W. Price and S. S. Pedro, editors, *Joint Proceedings of the Workshops at the International Conference on Educational Data Mining 2021 (EDM 2021)*, 2021.

[34] J. Wong, M. Khalil, M. Baars, B. D. Koning, and F. Paas. Exploring sequences of learner activities in relation to self-regulated learning in a massive open online course. *Computers & Education*, 2019.

[35] M. B. Zafar, I. Valera, M. Gomez-Rodriguez, and K. P. Gummadi. Fairness constraints: A flexible approach for fair classification. *Journal of Machine Learning Research*, 20(75):1–42, 2019.

[36] W. Zhao, J. Xia, X. Jiang, and T. He. A novel framework for deep knowledge tracing via gating-controlled forgetting and learning mechanisms. *Information Processing and Management*, 60(1):103114, 2023.