Diagnostic Questions: The NeurIPS 2020 Education Challenge

Zichao Wang [1] *, Angus Lamb[4*], Evgeny Saveliev[4], Pashmina Cameron[4], Yordan Zaykov[4], José Miguel Hernández-Lobato[245], Richard E. Turner[24], Richard G. Baraniuk[1], Craig Barton[3], Simon Peyton Jones[4], Simon Woodhead[3†], Cheng Zhang[4†]

Abstract

Digital technologies are becoming increasingly prevalent in education, enabling personalized, high quality education resources to be accessible by students across the world. Importantly, among these resources are *diagnostic questions*: the answers that the students give to these questions reveal key information about the specific nature of misconceptions that the students may hold. Analyzing the massive quantities of data stemming from students' interactions with these diagnostic questions can help us more accurately understand the students' learning status and thus allow us to automate learning curriculum recommendations. In this competition, participants will focus on the students' answer records to these multiple-choice diagnostic questions, with the aim of 1) accurately predicting which answers the students provide; 2) accurately predicting which questions have high quality; and 3) determining a personalized sequence of questions for each student that best predicts the student's answers. These tasks closely mimic the goals of a real-world educational platform and are highly representative of the educational challenges faced today. We provide over 20 million examples of students' answers to mathematics questions from Eedi, a leading educational platform which thousands of students interact with daily around the globe. Participants to this competition have a chance to make a lasting, real-world impact on the quality of personalized education for millions of students across the world.

Keywords

Personalized Education, Unsupervised Learning, Missing value prediction, Active Learning

*Equal contribution, co-first authors † co-senior authors [1]Rice University, [2]University of Cambridge, [3]Eedi, [4]Microsoft Research, [5]Alan Turing Institute.

Contents

1 Introduction

1.1 Background and impact

Background The prevalence of free or affordable online education systems is making high quality education available to a wider audience. On these platforms students can learn by watching instructional videos, reading (possibly interactive) course materials, and talking with other students and mentors in learning forums. To measure student understanding many of these platforms include an assessment component. By mining the data collected by these assessments, we can, in theory, extract useful educational information such as how students are learning, and recommend suitable learning interventions to improve learning outcomes. However, the quality of the insights derived is dependent on the quality of the questions in the assessments.

Formative assessment is concerned with the careful design of assessments which elicit detailed information that can be used to improve instruction and student learning while it is happening. A deceptively simply but powerful question type used for formative assessment is a *diagnostic question*.

A diagnostic question is a multiple-choice question with four answers, exactly one of which is correct and where each of the three incorrect answers is chosen to highlight a common misconception. If a student gets a diagnostic question wrong, educators are not left to guess why. The student's choice of incorrect answer reveals something about the nature of their misconception which is valuable information in seeking to help them resolve it [12]. Diagnostic questions can be constructed to induce retrieval of information pertaining to the incorrect alternatives. Therefore, students are not only challenged to consider why

the right answer is right, but why the wrong answers are wrong [7].

It is challenging to write good diagnostic questions, where each of the incorrect answers is a plausible distractor. Even experienced teachers find the task time-consuming. To address this challenge, a platform was created (`https://diagnosticquestions.com/`) by the team behind Eedi (`https://eedi.com`), to facilitate teachers crowd-sourcing diagnostic questions. Inevitably, there is variation in the quality of the questions created.

When teachers create diagnostic questions, each incorrect answer should be chosen to highlight a common misconception, but these misconceptions are not labelled or linked between questions. It is entirely possible for an incorrect answer to be chosen so poorly that it is obviously wrong, and therefore it will never be chosen by a student.

In order to diagnose student learning accurately it is essential that they are presented with good questions. A good diagnostic question identifies the specific nature of a students misconception. They need to be unambiguous, and crucially students should not be able to get them correct whilst still holding a key misconception. Moreover, teaching and learning time is limited, so we need to prioritise those questions which capture the most information about the student's knowledge and misconceptions.

Summary of competition and impact on education In this competition, we challenge participants to develop novel methodologies to understand and improve students learning and measure the quality of diagnostic questions. There are four tasks, described in Section 3 and summarized here:

1. The first task is to predict whether or not students will answer questions correctly.
   Real-world impact: Enable recommending questions of an appropriate difficulty to a given student that best fit their background and learning status.
2. The second task extends this to the prediction of which answer students choose for each question.
   Real-world impact: Enable discovering potential common misconceptions that students have by clustering of question-answer pairs which may indicate the same or related misconceptions.
3. The third task is to devise a metric to measure the quality of the questions. This metric will be evaluated against the opinions of domain experts.
   Real-world impact: Enable feedback to be provided to authors of diagnostic questions so they can revise poor quality questions and to guide teachers to choose questions for their students.
4. The fourth task is to acquire a limited set of answers from students for student performance prediction on unseen questions. This requires personalized machine

learning methods with estimation of value of information.
Real-world impact: Enable personalized assessments for each student to improve learning outcomes.

The competition provides an in-depth introduction to educational data mining because our tasks mimic the learning analytics and personalization tasks common to many education platforms. The competition will use data from an educational platform that is already deployed and used at scale. The data describes real answers given by real students to real questions. By providing the opportunity to work on genuine educational data and real problems in an engaging manner, our competition will attract talent to the important field of machine learning in education.

We expect the competition to bring fundamental advances to educational data mining technologies, particularly those that analyze students' learning progress and recommend personalized learning curricula. These methods will be deployed in a real educational platform where they will improve the learning outcomes of millions of students.

Impact on the machine learning community The competition involves multiple fundamental machine learning challenges that need to be addressed. Some challenges are common in recommender systems but appear in the context of educational data mining: how to deal with the sparsity of the data because each student answered only a small fraction of all questions? How to effectively use student and question metadata, such as student demographics, to improve the prediction? Other challenges are reminiscent of active learning: how to optimally select the sequence of questions in order to maximize prediction accuracy? Another challenge is how to effectively perform matrix completion for unordered, categorical data. Tackling these challenges in the unique context of educational data mining will be of significant technical interest to the NeurIPS community and, more broadly, the machine learning community as a whole.

1.2 Navigating the Competition

This document serves both as an introduction to the competition, and as a guide for participants to navigate the competition. **The remainder of this document contains important information on various aspect of the competition**. We encourage participants to become familiar with, and regularly refer back to, the following information during the competition:

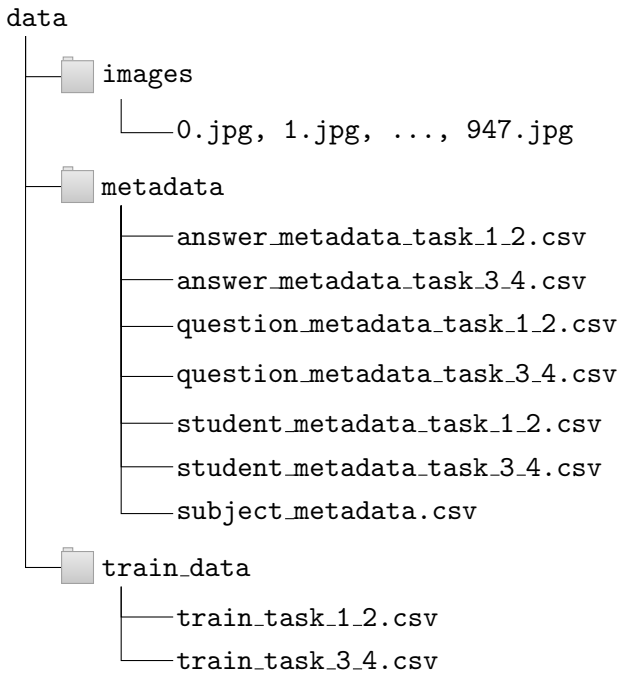
- A master list of files that participants will receive and where to download them (Section 1.3);

- Detailed descriptions of the data files (Section 2);
- Detailed descriptions of each task in the competition including evaluation metric(s) for each task (Section 3);
- Instructions for submitting each task to CodaLab (Section 4);
- A getting started guide providing scripts to load the data, perform local evaluation and prepare sample submissions (Section 5).

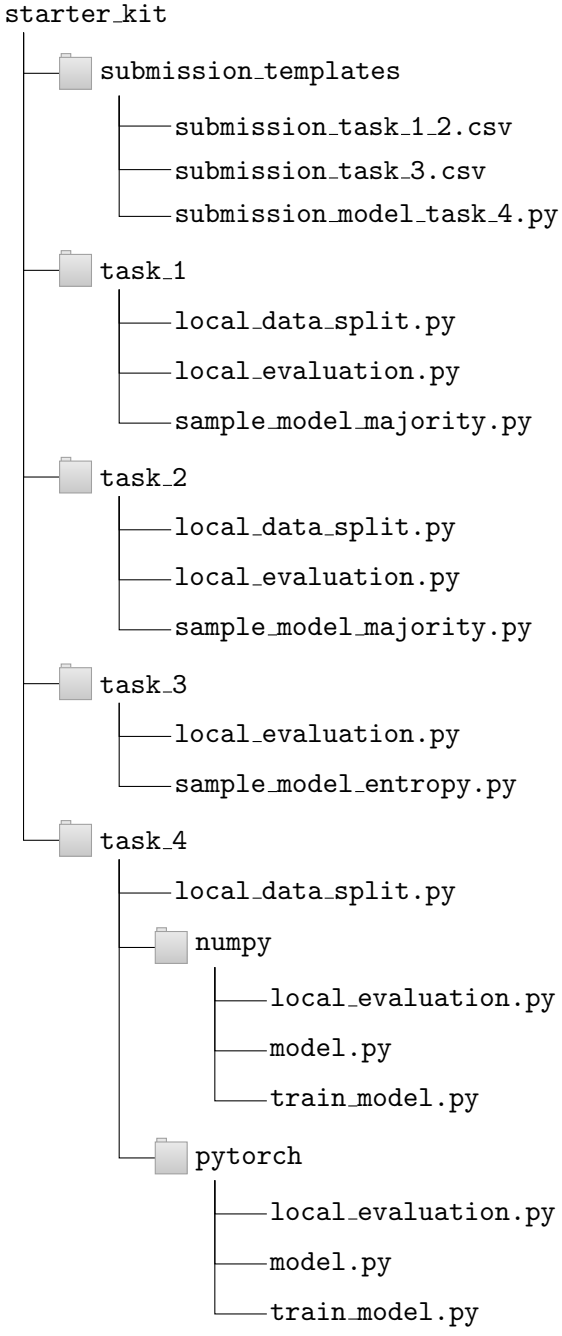
1.3 Competition File List

Below we provide a list of the files available to participants. The files below are in two zip files, both of which can be found in the CodaLab competition website. Click on the `Participate` tab, then `Get Data`.

Public Data Contains the data folder which contains three sub-folders (details of each file in Section 2):



Starting Kit Contains the folder `starter_kit` to help participants get started on data loading, local evaluation and submission preparation:



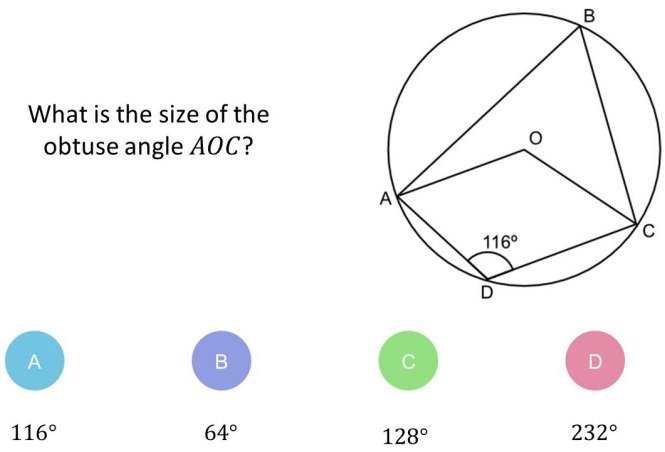


Figure 1: An example question from the education platform where the data we analyze is collected.

2 Data

Table 1: Example data.

QuestionId	UserId	AnswerId	AnswerValue	CorrectAnswer	IsCorrect
10322	452	8466	4	4	1
2955	11235	1592	3	2	0
3287	18545	1411	1	0	0
10322	13898	6950	2	1	0

We provide an extensive data provided by Eedi, an online education provider currently used in tens of thousands of schools, detailing student responses to multiple-choice diagnostic questions provided between September 2018 to May 2020. This platform offers crowd-sourced diagnostic questions to students from primary to high school (roughly between 7 and 18 years old). Each diagnostic question is a multiple-choice questions with 4 possible answer choices, exactly one of which is correct. Currently, the platform mainly focuses on mathematics questions. Figure 1 shows an example question from the platform. All data is available to download from the competition homepage on CodaLab; see Section 1.3.

The competition is split into 4 tasks: tasks 1 and 2 share a dataset, as do tasks 3 and 4. These datasets are largely identical in format, but use disjoint sets of questions.

All QuestionIds, UserIds and AnswerIds have been anonymized and have no discernable relation to those found in the product. Note that all such IDs for tasks 1 and 2 are anonymized separately from those for tasks 3 and 4: **IMPORTANT: Question, User and Answer IDs should not be linked between the data for these pairs of tasks!**. This is by design, to ensure that the two datasets are both self-contained.

2.1 Primary Data

This is main training data, consisting of records of answers given to questions by students. It can be found in the files `train_task_1_2.csv` and `train_task_3_4.csv` The columns are as follows:

- **QuestionId:** ID of the question answered.
- **UserId:** ID of the student who answered the question.
- **AnswerId:** Unique identifier for the (QuestionId, UserId) pair, used to join with associated answer metadata (see below).
- **IsCorrect:** Binary indicator for whether the student's answer was correct (1 is correct, 0 is incorrect).
- **CorrectAnswer:** The correct answer to the multiple-choice question (value in [1,2,3,4]).
- **AnswerValue:** The student's answer to the multiple-choice question (value in [1,2,3,4]).

Table 1 is an illustration of four data records in this format. Each student has typically answered only a tiny fraction of all possible questions and hence the matrix is extremely sparse. For tasks 1 and 2, we removed questions that have received fewer than 50 answers and students who have answered fewer than 50 questions. For tasks 3 and 4, where we are interested in a fixed set of questions, we removed all students who had answered fewer than 50 of these questions. In addition, when a student has multiple answer records to the same question, we keep the latest answer record. The data can be transformed into matrix form, where each row represents a student and each column represents a question.

For tasks 1 and 2, the individual answer records are randomly split into 80%/10%/10% training/public test/private test sets. For tasks 3 and 4, the UserIds are randomly split into 80%/10%/10% training/public test/private test sets. These preprocessing steps lead to training datasets of the following sizes:

- Tasks 1 and 2: 27613 questions, 118971 students, 15867850 answers

- Tasks 3 and 4: 948 questions, 4918 students, 1382727 answers

The total number of answer records these training sets exceeds 17 million, rendering manual analysis impractical and necessitating a data-driven, machine learning approach. For an illustration of the matrix representation of the data, see Figures 2 and 3 in Section 3.

2.2 Question Metadata

We provide the following metadata about each question:

- **SubjectId** Each subject covers an area of mathematics, at varying degrees of granularity. We provide IDs for each topic associated with a question in a list. Example topics could include "Algebra", "Data and Statistics", and "Geometry and Measure". These subjects are arranged in a tree structure, so that for instance "Factorising" is the parent subject of "Factorising into a Single Bracket". We provide details of this tree in an additional file `subject_metadata.csv` which contains the subject name and tree level associated with each SubjectId, in addition to the SubjectId of its parent subject.
- **Question content:** In Tasks 3 and 4, in addition to the topics, we will also provide the image presented to the student for each question, as shown in Figure 1, for each of the questions. The question images have been shared solely for the purpose of this competition and must not be used for any other purpose. The question images must not be printed or shared with anyone outside of the competition. The question wording is contained in the images but will not be made available as text.

2.3 Student Metadata

The following metadata is provided about students in the dataset:

- **UserId:** An ID uniquely identifying the student, which can be joined to the primary dataset.
- **Gender:** The student's gender, when available. 0 is unspecified, 1 is female, 2 is male and 3 is other.
- **DateOfBirth:** The student's date of birth, rounded to the first of the month.
- **PremiumPupil:** Whether the student is eligible for free school meals or pupil premium due to being financially disadvantaged.

2.4 Answer Metadata

The following metadata is provided about each individual answer record in the dataset:

- **UserId:** An ID uniquely identifying the answer, which can be joined to the primary dataset.
- **DateAnswered:** Time and date that the question was answered, to the nearest minute.
- **Confidence:** Percentage confidence score given for the answer. 0 means a random guess, 100 means total confidence.
- **GroupId:** The class (group of students) in which the student was assigned the question.
- **QuizId:** The assigned quiz which contains the question the student answered.
- **SchemeOfWorkId:** The scheme of work in which the student was assigned the question.

3 Task Details

In this section, we introduce the competition tasks and the evaluation metrics.

The competition consists of four tasks of varying styles. The first two tasks aim to predict the student's responses to every question in the dataset. These two tasks can be formulated in several ways, for instance as a recommender system challenge [5, 2, 3] or as a missing value imputation challenge [8, 13, 10, 4]. Here, each student only answers a small fraction of the questions while student responses to other questions are of great interest for personalized education. The third task focuses on evaluating question quality which is essential and remains an open question in the education domain [11]. The final task directly addresses the challenge of personalized education where personalized dynamic decision making [1, 6, 9] is needed. The first two tasks could serve as a useful basis on which to build solutions for the latter two tasks, but it is not necessary to take this approach. Participants are free to submit solutions to whichever tasks they wish and in whichever order they wish.

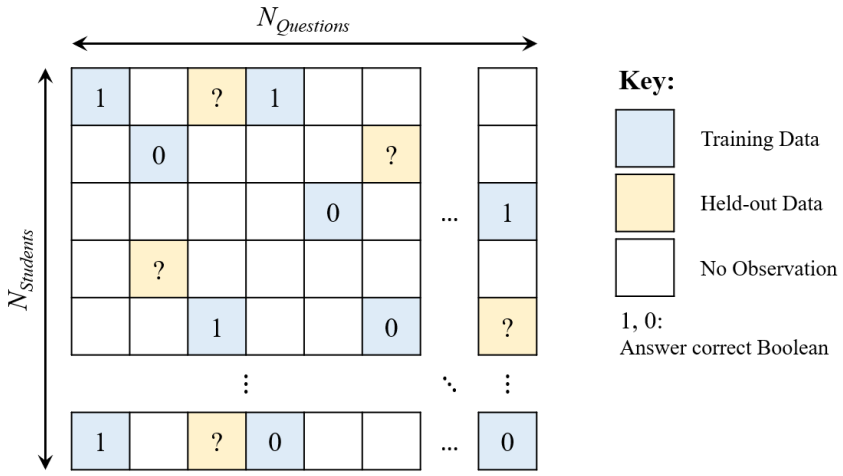


Figure 2: An illustration of the sparse matrix representation of the data for Task 1.

3.1 Task 1: Predict Student Responses – Right or Wrong

The first task is to predict whether a student answers a question correctly. The primary data used for this task is a table of records (`Student ID`, `Question ID`, `Is Correct` indicator) where the last column is binary valued. A sparse matrix representation of this data is illustrated in Figure 2. The participants are asked to predict the correctness indicator on a held-out subset of the students' answers. More specifically, for each student, a portion of the available records will be held out as the hidden test set on which evaluation will be performed.

Predicting the correctness of a student's answers to not yet answered (or newly introduced) questions is crucial for estimating the student's ability level in a real-life personalized education platform, and forms the groundwork for more advanced tasks. This task falls under the class of matrix completion, and is reminiscent of challenges often seen in the recommender systems domain in the case of binary data. Popular approaches in this domain such as matrix factorization or nearest-neighbour based methods may prove effective at this task.

3.1.1 Evaluation Metric

We use **prediction accuracy** as the metric, i.e., the number of predictions that match the true correctness indicator, divided by the total number of predictions (in the held-out

test set):

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{total predictions}}$$

3.2 Task 2: Predict Student Responses – Answer Prediction:

The second task is to predict which answer a student gave to a particular question. The primary data used for this task is a table of records (`StudentId`, `QuestionId`, `AnswerValue`, `CorrectAnswer`) where the last 2 columns are categorical taking values in [1, 2, 3, 4] (corresponding respectively to multiple-choice answer options A, B, C and D). The sparse matrix representation shown for Task 1 thus now looks as in Figure 3. The questions in our dataset are all multiple-choice, each with 4 potential choices and 1 correct choice, so this task is a multi-class prediction problem – the participants are asked to predict students' responses for a hidden, held-out subset of (StudentId, QuestionId) pairs.

Predicting the actual multiple-choice option for a student's answer allows analysing likely common misconceptions that a student may hold on a topic, and can thus form the basis for personalized advice and guidance on a real-life education platform. Clusters of question-answer pairs which are highly correlated may indicate that they correspond to the same, or related misconceptions. Understanding the relationships between misconceptions is a crucial problem to solve for curriculum development, it may inform the way a topic is taught and the sequencing of topics.

As in Task 1, this is a matrix completion task, but this time with unordered categorical data. Data of this type is rare in the recommender systems domain, where responses will typically be binary or ordinal (e.g. 1-5 stars), and so more novel approaches may be required in order to correctly predict students' answers and accurately model their misconceptions. The first two tasks form the foundation of analyzing students' learning, because most models that aim to produce such analytics rely on accurately modeling students and hence modelling their answers. Thus, participants competing in these two tasks are exposed to the fundamental task in educational data mining.

3.2.1 Evaluation Metric

We use the same metric **prediction accuracy** as above, except that the true answers are now categorical instead of binary.

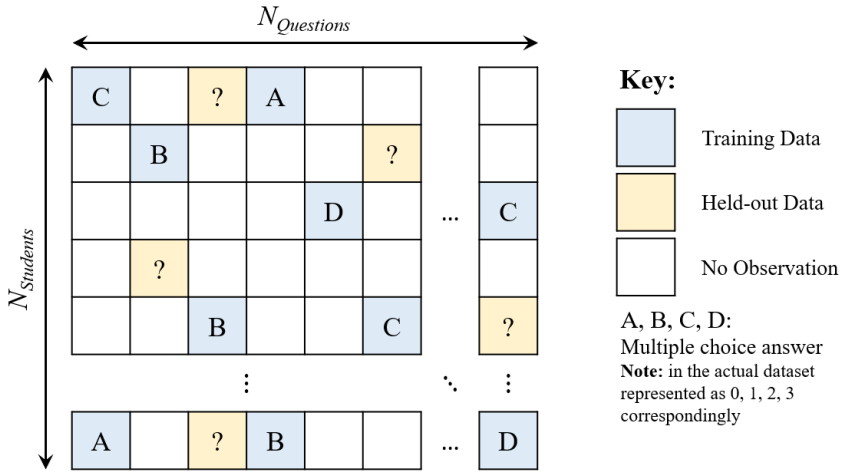


Figure 3: An illustration of the sparse matrix representation of the data for Task 2.

3.3 Task 3: Global Question Quality Assessment

The third task is to predict the "quality" of a question, as defined by a panel of domain experts (experienced teachers), based on the information learned from the students' answers found in the dataset. This task requires the definition of a metric for evaluating the question quality that mimics the experts' judgement of the question quality. Crucially, the experts' judgements will not be provided to the competition participants. The task is therefore very different in nature from the previous two, and is an unsupervised learning problem, demanding some innovative thinking.

In order to guide the participants, a note on how expert teachers judge question quality, including their intuition and the criteria they use, will be included as a supplementary material available to the participants. The participants may use this material to design their automatic question quality metrics. The content of this material will include an example of a prompt used in the expert data collection: Figure 4. In addition, the following "Golden rules" of quality question design have been identified by one of the domain experts, Craig Barton[1]:

- They should be clear and unambiguous
- They should test a single skill/concept
- Students should be able to answer them in less than 10 seconds

[1] `https://medium.com/eedi/what-makes-a-good-diagnostic-question-b760a65e0320`

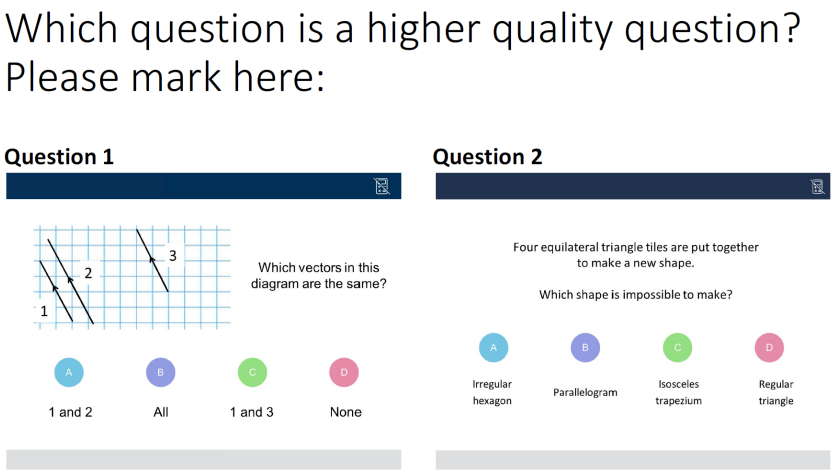


Figure 4: Example of a prompt used in collecting experts' judgement of pairwise relative question quality. In addition to this, the experts receive the following instructions: *On each of the following slides, you will see 2 questions, one on the left and one on the right. Please decide which question is of higher quality; ties are not allowed.*

- You should learn something from each incorrect response without the student needing to explain
- It is not possible to answer the question correctly whilst still holding a key misconception.

The question quality metric designed in this task is of paramount importance in crowd-sourced education applications, as it provides a scalable way to evaluate the quality of the questions submitted by teachers. The quality of crowd-sourced questions reflects directly on the usefulness of the platform to the students and teachers. The quality judgement can also be used for personalized guidance for the teachers, helping them to improve question quality.

The competitors are encouraged to utilise the machine learning model(s) used in the previous task(s) in defining this metric. Based on this metric, the participants must provide a ranking: rank from 1 to $N_{Questions}$, each uniquely mapping to a `Question ID` in the dataset. Rank 1 should correspond to the highest quality question, and so on, in order of decreasing question quality. The absolute values of the metric are not required. For an illustration of the required output see Figure 5. The evaluation procedure and metric are described in Section 1.5. swThere is no 1.5

This task can be viewed as an unsupervised learning task, as there is no explicit super-

Example submission:

Quality ranking	*Question ID*
1	Q-2748
2	Q-124
3	Q-3915
⋮	⋮
5,125	Q-10029
⋮	⋮
13,369	Q-6715

Scoring procedure:

Question-pair sampled for quality comparison	*Submission ranking implies*	*Expert judgement of quality*					
		Expert A		*Expert B*		*Expert C*	
Q-124 ≷ Q-10029	>	>	1	>	1	>	1
Q-11092 ≷ Q-3999	<	>	0	>	0	>	0
Q-844 ≷ Q-7491	<	<	1	<	1	<	1
Q-2748 ≷ Q-9882	>	>	1	<	0	<	0
Q-13001 ≷ Q-9115	>	>	1	>	1	<	0
Agreement with expert:		**0.80**		0.60		0.40	
Max agreement (task score):		**0.80** (Expert A)					

Key:

- ≷ Compare quality between question on the left (L) and question on the left (R)
- > Question L greater quality than question R
- < Question R greater quality than question L
- 1 Submission matches expert judgement
- 0 Submission doesn't match expert judgement

Figure 5: An illustration of the scoring process in Task 3. *Left*: The expected format of the submissions for Task 3 - a ranking of question quality over the `Question IDs`, in the decreasing order of quality. *Right*: An illustration of the performance metric calculation, see the steps in the main body text (Section 3.3.1). This example uses 5 question-pairs and 3 experts.

vision label available for question quality. Insights from areas such as information theory, feature selection and learning to rank may be relevant to this task.

3.3.1 Evaluation Metric

The participants will submit a quality ranking for the questions (Figure 5 *Left*). An unseen set of question-pairs will then be used to evaluate the quality of this ranking (and thus the underlying metric). Note that we have collected the experts' judgement of which question in each pair is of higher quality. The evaluation steps are then as follows (see also Figure 5 *Right* for an illustration):

- Determine, based on the submitted ranking, which question in each pair is of higher quality.
- Compare this with each expert's judgement (assign 1 if matching, 0 if not matching).
- For each expert i, determine the agreement fraction: $A_i = \frac{N_{matching\_pairs}}{N_{total\_pairs}}$.
- Find the *maximum* of these agreement fractions $A_{max} = \max_i A_i$. This will be used as the final evaluation metric for this task.

We are looking for metrics that can approximate *an* expert judgement really well, hence we use the maximum of the agreement fractions, rather than a mean of the agreement

fractions over all experts. The reasoning for this approach is that the quality metrics of the experts are in themselves subjective, and it is interesting to find whether a particular expert's approach can be approximated especially well by the use of machine learning.

3.4 Task 4: Personalized Questions

The fourth task is to interactively generate a sequence of questions to ask a student in order to maximise the predictive accuracy of a model on their remaining answers. Specifically, a participant's model will be provided with a set of previously-unseen students, whose answers to questions are completely hidden, and a set of potential questions to query for each student. The model will then choose a personalized question to query for each of these students in turn, and then their corresponding answer will be revealed to the model. Based on this information, the model should choose a second question to query for each student, and so on, until 10 questions have been asked in total.

The aim of the task is to maximise the predictive accuracy of a participant's model on a held-out set of questions for each student, after the model has been exposed to the 10 answers from each student. This task is of fundamental importance to personalized education, where we wish to accurately diagnose a student's level of understanding of various concepts while asking the minimum number of questions possible, in order to make the most efficient use of both student and teacher time. The task is also a crucial machine learning challenge, requiring participants to reason effectively about their model's uncertainty, and to use data as efficiently as possible.

This task can be viewed through the lens of a number of related fields, including active learning, reinforcement learning, bandit algorithms, Bayesian experimental design and Bayesian optimization, and insights drawn from any of these fields will likely prove useful.

3.4.1 Evaluation Metric

Submitted models will be asked to sequentially choose 10 query questions for every student in a held-out set of students. After each selection step, both the categorical answer and binary correctness indicator for these student-question pairs will be revealed to the model in private. The model is then given the opportunity to incorporate this new data or retrain after each question. After receiving 10 answers for each student, the model will be assessed on its prediction accuracy for predicting the binary correctness indicators for a held-out test set of answers for each of these students, that cannot be queried.

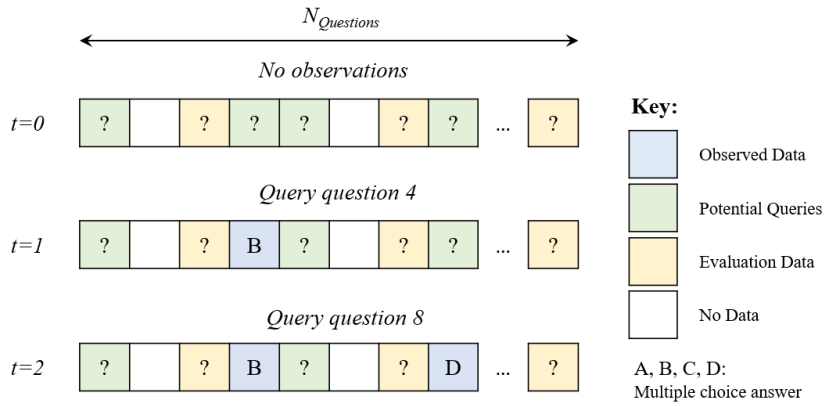


Figure 6: An illustration of the procedure for Task 4. On each time step, the model is able to train on the data in blue, and its predictive performance is assessed on the held-out data in yellow. The algorithm must then choose the next question to query from the set of green questions using this new model.

4 Submission Protocol

Each task contains two phases: a public evaluation phase and a private evaluation phase. Results in the public evaluation phase are displayed on a public leaderboard allowing participants to see how their submissions perform compared to other participants. Results in the private evaluation phase are hidden until the end of the competition. **Important: For each task, participants must submit to both the public and private evaluation phases separately. Submissions made solely to the public evaluation phase will not be used in the final judging of the competition. It is the participants' responsibility to make sure their submission to the private phase of each task represent their best results.** For tasks 1-3, submission template files are provided for both the public and private phases, which must be submitted to CodaLab. For task 4, a participants model must be submitted to both the public and private leaderboards, where it will be evaluated separately.

Before submission, the participants must first zip their submission files into one zip file. CodaLab only accepts one single `.zip` file as the submission file. The submission buttons are under `Participate` tab on the top of the competition website and under the `Submit/View Results` tab on the left. The tabs on the top of this page, e.g., `Task 1 Public`, contains the submission link to each of the tasks.

Below are the detailed submission instructions for each task.

4.1 Submission for Tasks 1-3

The first three tasks are evaluated through the submission of a zipped CSV file to Codalab:

- **Task 1**: Participants are provided with a CSV file (in the folder `starter_kit/submission_templates`) containing (UserId, QuestionId) pairs for which the answer is unseen. They must fill in their prediction for whether the student will answer the question correctly in each case. **Important: the submission file must be a `.zip` file containing a file named `submission_task_1.csv`.**
- **Task 2**: Participants are provided with a CSV file (in the folder `starter_kit/submission_templates`) containing (UserId, QuestionId) pairs for which the answer is unseen. They must fill in their prediction for which answer the student will give to the question in each case. **Important: the submission file must be a `.zip` file containing a file named `submission_task_2.csv`.**
- **Task 3**: Participants are provided with a CSV file (in the folder `starter_kit/submission_templates`) containing a list of all of the QuestionIds provided in the dataset for Task 3. They are asked to fill in a 'Ranking' column indicating the rank (1-948) they give to each question, where 1 is the highest-quality question and 948 is the lowest-quality question. **Important: the submission file must be a `.zip` file containing a file named `submission_task_3.csv`.**

4.2 Submission for Task 4

Task 4 is evaluated through a code submission. Participants are provided with a template file `submission_model_task_4.py`, which provides a simple API wrapper through which the evaluation script will interface with submitted models. The methods which participant must implement are as follows:

- **__init__**: Load a participant's model from this file or a neighbouring file (e.g. a separate `model.py` file where the model is defined) and perform any initialisation that is required.
- **select_questions**: Select the next question to query for each question, given the data observed by the model so far (both the binary correctness indicators and the specific multiple-choice answers) and an array indicating which questions can be queried for each student.
- **update_model**: Optionally update the model based on the new data revealed after revealing new answers.

- **predict**: Predict a binary correctness indicator for each student, for all questions for which we have not observed the student's answer.

Details of the specific signatures of these functions can be found in the template `submission_model_task_4.py` file. The evaluation procedure is as follows:

1. Initialise the model with `__init__()`.
2. For 10 steps, first select a new feature with `select_questions()`, reveal the selected values, and pass the new data to `update_model()`.
3. After 10 feature selection steps, call `predict()` to make binary predictions for the held-out target elements and evaluate the model's prediction accuracy.

The submission files for this task must contain `submission_model_task_4.py`. Users are free to include additional files, such as model definitions or trained model weights, in their submissions. To submit, the `submission_model_task_4.py` template file and any additional files should be zipped into one single `.zip` file and then submitted to CodaLab. **Important: please *do not* change the class method names in `submission_model_task_4.py`. Also, for any saved model files, participants *must* use `model_task_4_` as the prefix of the file name for submission**. For example, if the model is called `my_pytorch.pt`, then for submission , this model file must be renamed to `model_task_4_my_pytorch.pt`.

During evaluation, all training data and metadata files included as part of task 4 will be added to the root of the submission directory, and submissions may make use of these files as they wish. Simply specifying the name of the dataset to load should be sufficient to load the files – for instance, the training data can be accessed by submitted models at the path "`./train_task_3_4.csv`". Note that users are expected to upload a trained model for submission, as time limits on the submission compute workers will likely render training prior to evaluation too time-consuming.

For each task, there is a daily and total submission limit specified on the relevant submission page on Codalab. Unsuccessful submissions due to errors will not count against this total. **Submissions must be made separately to both the public and private components of the leaderboard**, with the final competition results being based solely on the private leaderboard.

4.3 Leaderboard

The submitted result will show up in a public leaderboard for each public phase of the competition. The public leaderboard shows how the participant(s) stand against other participants on the public evaluation data for each task.

Note that the leaderboard only shows the same "score" column for all tasks; participants should be aware that the meaning of the "score" is different for each task; please refer to the details of the evaluation metrics for each task in Section 3. Nevertheless, a more detailed result can be accessed in the "detailed results" part on the rightmost column of the table under the "Results" tab of the competition website.

For all private phases of the competition, no public leaderboard will be shown and results are only visible by the competition organizers.

4.4 Computation Environment

The evaluation on CodaLab is performed using the an off-the-shelf docker image that contains most of the data science, machine learning and deep learning packages. Participants must ensure that their submissions are able to run in this environment. Please see `https://github.com/ufoym/deepo` for more detail.

5 Getting Started: Sample Model, Local Evaluation and Submission Preparation

5.1 Quick Start

Both the public data and starter kit for the competition can be found under the tab `Participate/Get Data/` from the competition homepage. The starter kit contains a number of utility scripts and sample models to allow easy participation in the competition. Submission templates for preparing submissions in the correct format for each task are included in the `submission_templates` directory.

To quickly get started with the competition, follow these instructions:

1. Download the training data and starter kit from the competition homepage.
2. Place the downloaded `data` directory into the root of the `starter_kit` directory.
3. (Optional) Run the `local_data_split.py` files for each task in order to generate validation sets for local model evaluation.
4. Run the sample model files provided for tasks 1-3 in order to generate sample submissions for the competition. Task 4 requires no generation, as participants must submit the model code itself.

To submit solutions to Tasks 1-3, participants should submit a `.zip` file containing a completed submission template file named `submission_task_n.csv` where `n` is the task number. This file should then be uploaded to **both** the public and private phases of the appropriate task.

To submit solutions to Task 4, participants should submit a `.zip` file containing a completed submission API wrapper file `submission_model_task_4.py`, in additional to any additional model files or artifacts required in order to run the trained model.

Each task has its own self-contained directory, which typically includes a script for creating a local "validation set" for local model evaluation, a script for performing local model evaluation, and an example model to help get started with the task.

Further details for each task's resources are provided in the following sections.

5.2 Task 1

Available Files The following files are available for Task 1:

- Training data: `train_task_1_2.csv`, in the folder `data/train_data`
- Submission template: `submission_task_1_2.csv`, in the folder `starter_kit/submission_templates`
- Question metadata: `question_metadata_task_1_2.csv`, in the folder `data/metadata`
- Student metadata: `student_metadata_task_1_2.csv`, in the folder `data/metadata`
- Answer metadata: `answer_metadata_task_1_2.csv`, in the folder `data/metadata`
- Local evaluation scripts: `local_data_split.py`, `local_evaluation.py`, in the folder `starter_kit/task_1`
- Sample baseline model: `sample_model_majority.py`, in the folder `starter_kit/task_1`

Submission to CodaLab To submit to CodaLab, participants must submit results containing predictions for *each and every* `(UserId, QuestionId)` pair in the submission template file `submission_task_1_2.csv`. Running the provided `sample_model_majority.py` script will generate an example of this file, creating a directory `../submissions` by default where the generated file is named as `submission_task_1.csv` which is ready to be zipped and submitted.

Important: make sure the name of the prediction column in submissions is `IsCorrect`.

Local Evaluation To evaluate locally on a validation set produced from the training data, follow the steps below:

1. Navigate into the folder `starter_kit/task_1`
2. Split the data. We have provided a script `local_data_split.py` to do so; participants can perform data split by running
   `python local_data_split.py`
   which generates a train and validation set using the training data. By default, the split files are named as `train_task_1_2.csv` and `valid_task_1_2.csv` and are saved in the folder `data/test_input`.
3. Run your model and make predictions. We have provided a sample model `sample_model_majority.py` to help get started. Note that in order to run this model on the local evaluation data split, the block of code marked "Default arguments for Codalab submission" should be commented out, and the block marked "Default arguments for local evaluation" should be uncommented. To get predictions using this model, please run
   `python sample_model_majority.py`
   This model outputs a `.csv` file containing the results at (`data/test_input/test_submission_task_1.csv`).
4. Evaluation. Run
   `python local_evaluation.py`
   with appropriate options; see `argparse` arguments. This command computes and saves the score using the prediction and validation set. A score and confusion matrix are saved to the `output_dir` which by default is `data/test_output`. The official evaluation for this task implements the same evaluation metric as that in `local_evaluation.py`.

5.3 Task 2

Available Files The following files are available for Task 2:

- Training data: `train_task_1_2.csv`, in the folder `data/train_data`
- Submission template: `submission_task_1_2.csv`, in the folder `starter_kit/submission_templates`
- Question metadata: `question_metadata_task_1_2.csv`, in the folder `data/metadata`
- Student metadata: `student_metadata_task_1_2.csv`, in the folder `data/metadata`

- Answer metadata: `answer_metadata_task_1_2.csv`, in the folder `data/metadata`
- Subject metadata: `subject_metadata.csv`, in the folder `data/metadata`
- Local evaluation scripts: `local_data_split.py`, `local_evaluation.py`, in the folder `starter_kit/task_2`
- Sample baseline model: `sample_model_majority.py`, in the folder `task_2`

Local Evaluation and Prepare Submission Since Task 2 is very similar in nature to Task 1, the training data and submission templates are the same and the local evaluation scripts are similar. The only difference is that Task 2 predicts the actual response that a student makes to a question (answer A, B, C or D to each multiple-choice question, encoded as 1, 2, 3 or 4 respectively).

Important: make sure the name of the prediction column in the submission file is `AnswerValue`.

5.4 Task 3

Available Files The following files are available for Task 3:

- Training data: `train_task_3_4.csv`, in the folder `data/train_data`
- Submission template: `submission_task_3.csv`, in the folder `starter_kit/submission_templates`
- Question images: in the folder `data/images/`
- Question metadata: `question_metadata_task_3_4.csv`, in the folder `data/metadata`
- Student metadata: `student_metadata_task_3_4.csv`, in the folder `data/metadata`
- Answer metadata: `answer_metadata_task_3_4.csv`, in the folder `data/metadata`
- Subject metadata: `subject_metadata.csv`, in the folder `data/metadata`
- Sample baseline model: `sample_model_entropy.py`, in the folder `starter_kit/task_3`

Submission to CodaLab This task asks participants to rank the quality of questions in the training data `train_task_3_4.csv` in descending order, i.e., rank 1 represents the highest quality, rank 2 represents the second highest quality, etc. There is no ground-truth provided and local evaluation is not possible; rather, we provide a baseline model which ranks the questions based on an estimation of their entropy (`sample_model_entropy.py`) to generate results appropriate for submission for Task 3. See the implementation in this files for details of the computations. The submission file contains 2 columns `(QuestionId, ranking)` where the second column is the ranking for each question. Each question should have a unique ranking, i.e., no two rankings should be the same.

The training data and metadata provided for tasks 3 and 4 is in the same format as that of tasks 1 and 2, but uses a disjoint, smaller set of questions. As explained in section 2, the randomized IDs used in Tasks 1 and 2 are generated independently of those used in Tasks 3 and 4, and so participants should not attempt to use the data from Tasks 1 and 2 to aid them in Tasks 3 and 4.

To prepare submission run your model which generates the ranking for all question IDs in the `submission_task_3.csv` file. The provided entropy-based model demonstrates this process; simply run

```
python sample_model_random.py
```

This will create a prediction file `test_submission_task_3.csv` in the `../submissions` folder by default. This file is ready to be zipped and submitted.

Important: the ranking column must be named `ranking` for the CodaLab evaluation script to correct read the participants' predicted quality rankings.

5.5 Task 4

Available Files The following files are available for Task 4:

- Training data: `train_task_3_4.csv`, in the folder `data/train_data`
- Submission template file: `submission_task_4.py`, in the folder `submission_templates`
- Question images: in the folder `data/images/`
- Question metadata: `question_metadata_task_3_4.csv`, in the folder `data/metadata`
- Student metadata: `student_metadata_task_3_4.csv`, in the folder `data/metadata`
- Answer metadata: `answer_metadata_task_3_4.csv`, in the folder `data/metadata`
- Subject metadata: `subject_metadata.csv`, in the folder `data/metadata`

- Local evaluation scripts and models:
  - `local_data_split.py` is in the folder `starter_kit/task_4`
  - `local_evaluation.py`, `train_model.py`, `submission_model_task_4.py`, `model_task_4.pt`, `model.py` in the folder `starter_kit/task_4/pytorch`
  - `local_evaluation.py`, `train_model.py`, `submission_model_task_4.py`, `model_task_4_most_popular.npy`, `model_task_4_num_answers.npy` `model.py`, in the folder `starter_kit/task_4/numpy`

Submission to CodaLab As detailed in Submission for Task 4, participants should submit to CodaLab a `.zip` file containing their completed `submission_model_task_4.py` file along with any supplementary model code or weights, where it will be evaluated in private. Submissions may assume that all training data and metadata files for the task will be included same directory as the submission files (i.e., `./`).

We have provided sample models and script to generate a sample submission ready for CodaLab. To do so, please follow the steps below:

1. Navigate to either `starter_kit/task_4/numpy` or `starter_kit/task_4/pytorch`
2. Run
   ```
   python train_model.py
   ```
   which saves a model file to either the NumPy or PyTorch folder.
3. If using the NumPy model, zip `model.py`, `submission_model_task_4.py`, `model_task_4_most_popular.npy` and `model_task_4_num_answers.npy` to a `.zip` file. If using the PyTorch model, zip `model.py`, `submission_model_task_4.py` and `model_task_4.pt` into a `.zip` file. Details on the naming of the files and what can be changed inside the submission template file `submission_model_task_4.py` are discussed in Section 4.2.

Local Evaluation In the starter kit for Task 4, a mock test environment is provided as part of the competition API, in order to allow the entrants to check that their submission will work with the evaluation procedure. Both the NumPy and PyTorch models described above can be evaluated locally in this way.

1. Navigate into the folder `starter_kit/task_4`
2. Split the data. We have provide a script `local_data_split.py` to do so; participants can perform data split by running
   ```
   python local_data_split.py
   ```
   which generates a train and validation set using the training data. By default, the

split files are named as `train_task_4.csv` and `valid_task_4.csv` and are saved in the folder `data/test_input`.

3. Train your model. We have provided a sample model file and training scripts using both NumPy and PyTorch. To train the sample models, navigate to either `starter_kit/task_4/numpy` or `starter_kit/task_4/pytorch` and run

   ```
   python train_model.py
   ```

   Note that in the NumPy model, the path to the training data in the `train_model()` method will need to updated to point to the newly created "local test" split. This script will save a model file to the NumPy or PyTorch folders above; see the `train_model.py` script for more details. Note that the provided PyTorch model class does not implement a method to train the model;

4. Evaluate. Run

   ```
   python local_evaluation.py
   ```

   This command computes and prints the sequence of selected questions and the final accuracy.

References

[1] Alekh Agarwal, Sarah Bird, Markus Cozowicz, Luong Hoang, John Langford, Stephen Lee, Jiaji Li, Dan Melamed, Gal Oshri, Oswaldo Ribas, et al. A multiworld testing decision service. *arXiv preprint arXiv:1606.03966*, 7, 2016.

[2] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[3] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup11. In *Proceedings of KDD Cup 2011*, pages 3–18, 2012.

[4] W. Gong, S. Tschiatschek, R. Turner, S. Nowozin, J. M. Hernández-Lobato, and C. Zhang. Icebreaker: Element-wise active information acquisition with bayesian deep latent gaussian model. In *Proc. Advances in Neural Information Processing Systems*, 2019.

[5] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[6] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[7] J. Little, E. Frickey, and A. Fung. The role of retrieval in answering multiple-choice questions. In *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 2018.

[8] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[9] C. Ma, S. Tschiatschek, K. Palla, J. M. Hernández-Lobato, S. Nowozin, and C. Zhang. EDDI: Efficient dynamic discovery of high-value information with partial VAE. In *Proc. International Conference on Machine Learning*, volume 97, pages 4234–4243, Jun. 2019.

[10] Daniel J Stekhoven and Peter Bühlmann. Missforestnon-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.

[11] Zichao Wang, Sebastian Tschiatschek, Simon Woodhead, José Miguel Hernández-Lobato, Simon Peyton Jones, and Cheng Zhang. Large-scale educational question analysis with partial variational auto-encoders. *arXiv preprint arXiv:2003.05980*, 2020.

[12] C. Wylie and D. Wiliam. Diagnostic questions: Is there value in just one? In *Paper presented at the annual meeting of the American Educational Research Association (AERA) and the National Council on Measurement in Education (NCME) held between April 6 to 12, 2006, in San Francisco, CA*, 2006.

[13] Jinsung Yoon, James Jordon, and Mihaela Van Der Schaar. Gain: Missing data imputation using generative adversarial nets. *arXiv preprint arXiv:1806.02920*, 2018.