A Constraints-Based Approach to Fully Interpretable Neural Networks for Detecting Learner Behaviors

Juan D. Pinto
University of Illinois Urbana-Champaign
jdpinto2@illinois.edu

Luc Paquette
University of Illinois Urbana-Champaign
lpaq@illinois.edu

ABSTRACT

The increasing use of complex machine learning models in education has led to concerns about their interpretability, which in turn has spurred interest in developing explainability techniques that are both faithful to the model's inner workings and intelligible to human end-users. In this paper, we describe a novel approach to creating a neural-network-based behavior detection model that is interpretable by design. Our model is fully interpretable, meaning that the parameters we extract for our explanations have a clear interpretation, fully capture the model's learned knowledge about the learner behavior of interest, and can be used to create explanations that are both faithful and intelligible. We achieve this by implementing a series of constraints to the model that both simplify its inference process and bring it closer to a human conception of the task at hand. We train the model to detect gaming-the-system behavior, evaluate its performance on this task, and compare its learned patterns to those identified by human experts. Our results show that the model is successfully able to learn patterns indicative of gaming-the-system behavior while providing evidence for fully interpretable explanations. We discuss the implications of our approach and suggest ways to evaluate explainability using a human-grounded approach.

Keywords

Explainable AI, model transparency, interpretable neural networks

1. INTRODUCTION

The field of education, as with many other areas of research, has continued to see an increasing use of complex machine learning models as these have become more powerful and versatile over the years. However, as these models have grown in complexity, they have also become more difficult to interpret, leading to concerns around fairness, accountability, and trust [8], while also obscuring pedagogical insights that could improve learning outcomes among students.

There has been significant interest in tackling these issues within the eXplainable AI (XAI) community, with a growing group of educational data mining (EDM) researchers focusing on the implications and possible solutions to the problems arising from highly complex models (as evidenced by workshops specializing on XAI in education [21]). There is growing awareness of the inherent limitations of post-hoc explainability techniques (i.e. generating explanations, typically feature importance measures, based on analyses of a model's inputs and outputs) which may often make them unsuitable for use in educational settings.

In this paper, we describe a novel approach to training a neural-network-based behavior detection model (more specifically, a convolutional neural network) that is interpretable by design [28]. Our model is *fully interpretable*, by which we mean that the parameters we extract for our explanations (1) have a clear interpretation, (2) fully capture the model's learned knowledge about the learner behavior of interest, and (3) can be used to create explanations that are both faithful and intelligible. We achieve this by implementing a series of constraints to the model that both simplify its inference process and bring it closer to a human conception of the task at hand.

We specifically focus on the detection of gaming-the-system behavior, a type of learner disengagement with an educational task. We chose this due to the existence of previous expert-based features and models [17] that serve as a baseline of both accuracy and interpretability. We also chose to use a deep learning model due to their rising popularity among EDM researchers and the difficulty of faithfully interpreting their decision-making process [19].

Our research is guided by the following questions:

1. Can we train a convolutional neural network to detect gaming-the-system behavior with convolutional filters acting as explicit behavioral patterns?
2. Can we alter the model's architecture so that the patterns are the only learnable parameters?
3. How can we create a differentiable perfect matching function that allows the model to definitively indicate a match or non-match based on the presence of any exact filter match to the input?
4. How do the patterns learned by our model compare to those identified by human experts?

5. How can we evaluate the interpretability of our model's explanations?

2. BACKGROUND

2.1 Intrinsically interpretable models

The literature abounds with descriptions of the differences between models that are intrinsically interpretable and those that are opaque (often referred to as "black-box models") [11, 10]. Yet fundamentally, any model's interpretability must be judged by the interpretability of the explanations derived from it [23, 20]. These explanations can be derived from one of two sources: the model's inner workings themselves (e.g. its parameters or gradients) or via post-hoc techniques that use simplified approximations based on the model's inputs and outputs alone (such as LIME [22] or SHAP [12]).

With this understanding, [23] has proposed defining an explanation as the output of an interpretation function being performed on a piece of evidence. Evidence, in this context, refers to direct aspects of the model's workings, which can be extracted from any combination of its parameters, gradients, inputs, and outputs. A piece of evidence in itself carries no semantic significance—this can only be added by an interpretation function. The interpretation function describes how the model makes use of the evidence. By this definition, an explanation is therefore an inference made from interpreting specific evidence.

An important concept that [23] introduce is that of *explanatory potential*. This is the extent to which a specific set of evidences accounts for the whole of the model's predictions. In other words, using a subset of a model's parameters to create an explanation may be sufficient to explain a portion of the transformations that the input goes through to reach the output, say 70%, but it may not provide a complete picture. Of course, a model does not typically use each parameter equally when making predictions, so the explanatory potential of each individual piece of evidence (along with their interactions) may vary.

It should be noted that explanatory potential as defined by [23] pertains only to a set of evidences. It is the upper limit of the extent to which an explanation derived from it can account for the model's predictions, but an explanation also depends on an accurate interpretation function. Interpretations can be particularly challenging to identify given sufficiently complex transformations. Furthermore, even when derived from sound interpretations and a set of evidences with high explanatory potential, an explanation may still prove to be too complex for a target audience to understand. Thus why creating useful explanations must ultimately be a human-centered endeavor.

With these considerations in mind, the question then arises of how to evaluate not just an explanation's *potential* explainability, but its actual usefulness in practice.

2.2 Evaluating explainability

Predictive and inferential models are typically evaluated primarily based on the accuracy of their outputs. Many robust methods for measuring different aspects of this accuracy have been devised and validated. It has become clear that no single accuracy metric captures the full complexity of a model's performance, and so a variety of metrics are used to provide a more complete picture [3, 18]. When it comes to evaluating explainability, however, there is yet no clear consensus.

This may be in part due to the subjective nature of explainability. While accuracy can be measured by comparing a model's predictions to a ground truth, explainability is a more abstract concept that depends on the needs and expectations of one or more end-users [26]. Still, some helpful frameworks have been proposed to guide the evaluation of explanations, both theoretical and practical.

2.2.1 Theoretical framework

[20] proposed a theoretical framework that brings together important elements to consider. They describe two main criteria through which explanations can be evaluated, with both as prerequisites for a *useful* explanation: *intelligibility* and *faithfulness*. Intelligibility refers to the ease with which a human can understand the explanation (also referred to as *comprehensibility* [4]), while faithfulness refers to the extent to which the explanation accurately reflects the model's inner workings. The authors argue that both criteria are necessary for a useful explanation, since an explanation that is not intelligible will likely not be used, and an explanation that is not faithful may be misleading.

As prerequisites to these two criteria, [20] further propose that an intelligible explanation must be *plausible* (i.e. it aligns with human intuition [6]) while a faithful explanation must be *stable* (i.e. it does not change drastically with small perturbations to the model's input [1]).

These criteria help explain why post-hoc explainability techniques, which are popular due to their ease of use and model-agnostic nature, are often problematic. Their faithfulness is difficult to measure with certainty since they rely exclusively on a model's inputs and outputs, with no direct access to its internal evidences, thus treating it precisely like a black box [24]. Furthermore, while individual post-hoc techniques may lead to internally stable predictions, different approaches often produce different explanations for the same model and inputs [9]. Their appeal is partly due to their ability to produce plausible and intelligible explanations—but when coupled with a lack of faithfulness, this may lead to problems where even experts are unable to decide which explanation to trust [27].

2.2.2 Practical framework

On the practical side of evaluation, [5] have proposed three methodological categories, as well as specific approaches, for evaluating explanations. Each evaluation approach emphasizes a different set of criteria [20].

First, application-grounded evaluation involves testing a model in the real world for the target application for which it was developed. Approaches in this category can be costly and time consuming but have high fidelity with the needs of end-users. For example, learning dashboards are sometimes evaluated based on how well they help instructors understand their students' learning and provide help [25, 29].

Such evaluation methods can effectively measure explanation intelligibility, but they do not directly measure faithfulness.

Second, human-grounded evaluation involves measuring how well humans can accurately answer questions about the model based on its explanation. Examples of the types of experiments in this category include: binary forced choice, where participants must pick which explanation they consider best when presented with multiple options (e.g. [27]); forward simulation, where participants must correctly predict the model's output given specific inputs; and counterfactual simulation, where participants must correctly identify how a specific input needs to be changed in order to alter the model's given output. The latter two approaches in particular can serve as rigorous tests of the faithfulness of an explanation, while simultaneously capturing many aspects of intelligibility.

Finally, functionally grounded evaluation involves measuring some abstract aspects of a model or its explanation that capture constructs related to interpretability, but without human involvement. Being the least direct category, such approaches make it difficult to make robust claims about either intelligibility or faithfulness, but some measures such as model sparsity or explanation simplicity [13] can be said to make proxy measures of intelligibility. They can likewise be used to measure stability.

The modeling approach we describe in this paper will particularly emphasize faithfulness. That is, we are interested in training a model that makes it possible to generate explanations that fully capture the model's inference process, all while not becoming overwhelming and thus remaining intelligible. This is what we refer to as *full interpretability*. For this reason, we also aim for explanations that can be evaluated using human-grounded methods, especially forward simulation and counterfactual simulation.

2.3 Interpretable gaming behavior detection

Gaming the system is a well-studied learner behavior in which learners exploit the properties of a learning environment in order to succeed at a task, often by guessing or extracting answers from a support system. Using sequences of five student-actions from Cognitive Tutor Algebra that were previously coded for gaming behavior by [2], [17] set out to create a classification model based on human expert insights.

Through cognitive task analysis—in which an expert in the behavior of interest reasons through various action sequences and explains their thinking out loud—[17] were able to identify a set of features designed to capture elements of students' problem-solving behavior that experts pay attention to when looking for gaming-the-system behavior. For example, these features included whether a student reuses the same answer in a different part of the problem interface or if the student enters consecutive similar answers. Each feature is binary, indicating whether it was observed or not for a given action step. An action step is a point in time at which data was collected, and is triggered by the student either making a submission or asking for a hint.

With iterative input from the expert, they then used these features to create a set of 13 patterns indicative of gaming-the-system behavior. Using these patterns as sliding windows

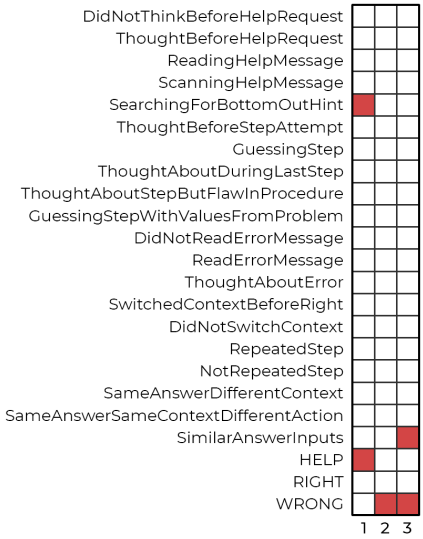

Figure 1: Example expert pattern from cognitive model.

of consecutive actions in each sequence—thus serving as a cognitive model—the researchers achieved a Cohen's kappa of 0.430 on the training data and 0.330 on unseen test data. Their model was able to outperform a machine learning model trained on the same data by [2] that achieved a kappa of 0.218 and AUC of 0.691 on held-out test data (as reported in [15]).

An example of one of these patterns can be seen in Figure 1. Each row is a different feature identified via cognitive task analysis, and each column indicates a student action step. The pattern is a 3-action sequence. As originally written out, this pattern is as follows:

> ***help*** *&* *[searching for bottom-out hint]* → ***incorrect*** → *[similar answer]* *&* ***incorrect***

[15] later trained a series of machine learning models using features that were automatically engineered using the results of the cognitive task analysis. Their best model, an ensemble of naive Bayes classifiers, achieved a kappa of 0.376 and AUC of 0.876 on the same test set. Importantly, they compared the interpretability of all three models trained and tested on the same data: the original ML model [2], the cognitive model [17], and the hybrid approach [15]. Not surprisingly, they found the cognitive model to be the most interpretable. The hybrid model that used expert-informed features provided an interesting middle ground, but the simple expert patterns that made up the cognitive model proved to be the most effective for human end-users. The ML model in this case was the least accurate and interpretable, but it also required far fewer resources to create.

An analysis of the generalizability of the same 13 expert patterns explored how their frequency varied across different student populations and learning environments [14]. They found that differences in the learning environments were more strongly associated with differences in gaming behavior

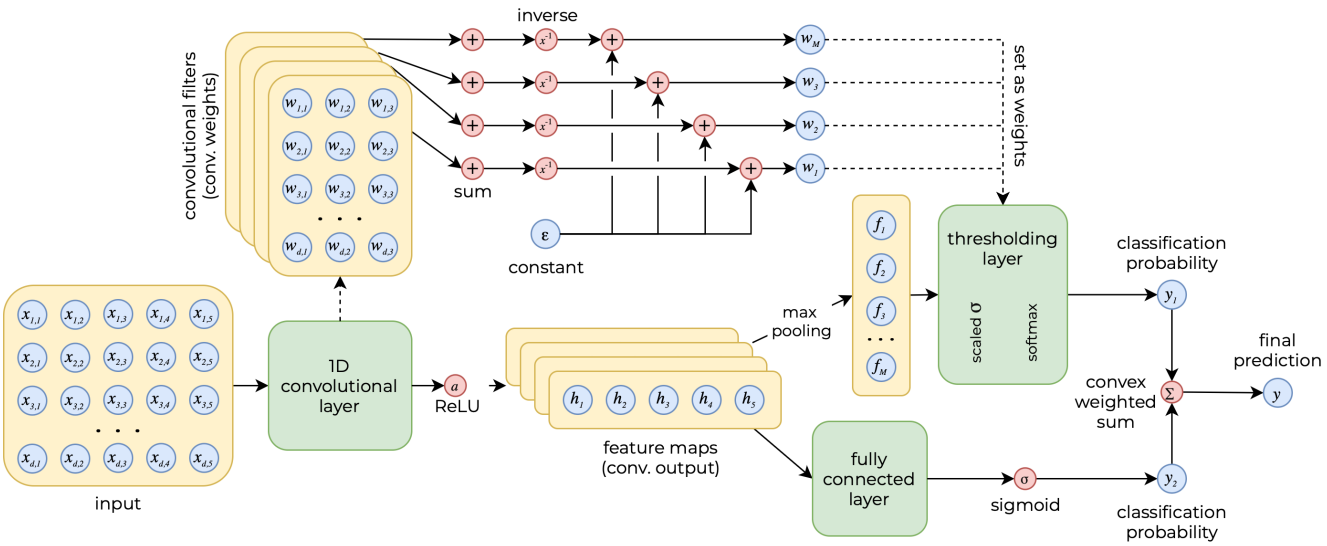

Figure 2: Model architecture.

than were student populations. A similar study found that the cognitive model generalized better across learning environments than the ML model [16]. This suggests that the expert patterns identified by cognitive task analysis, while (not surprisingly) affected by aspects of the data from which they were generated, are nevertheless robust and can be used across contexts.

3. METHODS

Rather than relying on post-hoc explainability techniques to create explanations with questionable faithfulness, we set out to create a neural-network-based behavior detection model that is interpretable by design [28]. Furthermore, we aimed to create a model that is *fully interpretable*. Such a model would be able to provide sufficient evidence to lend 100% explanatory potential while ensuring that this evidence would have a clear interpretation. We deemed these qualities to be prerequisites to creating effective explanations that are both entirely faithful to the model's inner workings and intelligible to human end-users.

For this task, we use the same dataset and features as [17] to train our model for two primary reasons. First, this allows us to directly compare the accuracy of our model with that of their cognitive model. Second, it gives us a template for the types of patterns we would like our model to be able to create. These patterns serve as the cognitive model's explanations—which are both fully faithful, since they make up the entirety of the cognitive model, and were manually designed by humans, suggesting a high level of intelligibility—so we reasoned that they would serve as a useful target for our own explanations. This also allows us to directly measure the similarities and differences between our model's learned patterns and those identified by human experts.

The dataset consists of sequences of actions from 59 students using the Cognitive Tutor Algebra system during an entire school year. Cognitive Tutor tasks students with solving multi-step mathematical problems and can provide on-demand hints. A total of 10,397 clips (i.e., student action sequences) were previously labeled by an expert [2] and contained 708 examples of gaming-the-system behavior (6.8%).

Our model is purposefully simple, consisting of a single convolutional layer followed by a novel layer that we refer to as a *differentiable thresholding fully connected layer* (explained in detail in section 3.2, see also Figure 2). The convolutional layer is designed to learn student patterns indicative of gaming-the-system behavior and which are similar in form to those identified by human experts in [17].

For the sake of simplicity, we only use student action sequences from the dataset that have a fixed length of 5 action steps (85% of all sequences). We use the same held-out test set as [17], consisting of 25% of the data. We further split the remaining data into a training/validation set using an 80/20 stratified split, using the validation set to tune hyperparameters and further refine our learned patterns. The final sets consist of 5,249 training clips, 1,313 validation clips, and 2,170 test clips, each with about 6% positively labeled instances.

Since convolutional filters must have a fixed kernel length (i.e. number of action steps), we use a kernel length of 3, corresponding with the most common pattern length identified by the human expert. Theoretically, the model could learn patterns with smaller lengths than this, but not longer. We set the convolutional layer's padding to 1 to allow the model to learn shorter patterns on the edges of input action sequences, and we set the number of filters to 2,048 to allow the model to learn a wide variety of patterns.

During training, the model contains additional elements, such as a dropout layer for the outputs of the convolutional layer, as well as a fully connected layer that acts as a traditional classifier via a weighted branching architecture, but these are not used during inference.

To achieve full interpretability, we followed two atypical approaches. First, our model architecture is minimalist—i.e. it avoids any learnable weights outside of the target patterns contained in the convolutional layer's filters, such as bias terms or fully-connected-layer weights. Second, we introduced a series of constraints to the model's learning process that encourage the convolutional weights to follow certain guidelines. Both of these approaches allow us to control the amount of inherent flexibility in the model.

Flexibility is key to models that aim to be interpretable by design. Closely related to the tradeoff between bias and variance, increased flexibility (which is itself related to decreased bias) tends to lead to more complex models that make interpretability more difficult. Thus, our constraints are designed to decrease flexibility just enough to allow the model to learn the patterns we are interested in, while still maintaining a high level of predictive accuracy.

3.1 Loss function constraints

The model was trained using binary cross entropy as its main loss term. To this were added four regularization terms, each aimed to constrain the model's learning in a specific way. The full loss function is given by:

$$L = L_{\text{BCE}} + \gamma_1 L_{\text{bin}} + \gamma_2 L_{\text{min}} + \gamma_3 L_{\text{sub}} + \gamma_4 L_{\text{poss}}$$

where L_{BCE} is the main loss term (binary cross entropy loss) and γ_i are scaling weights that control the impact of each regularization term.

All of these constraints specifically affect the weights in the model's convolutional layer. Figure 3 visually demonstrates the impact of each additional constraint on the convolutional filters. We will now describe each of these regularization terms in turn.

3.1.1 *Binarize convolutional filter weights*

The first regularization term constrains the model's convolutional layer to learn weights very close to 0 or 1 by penalizing values that deviate from these. Constraining the weights in this way enables a more straight-forward interpretation of each filter as a sequential pattern that emulates the binary presence or absence of specific features per action step. This binarizing effect can be seen in the difference between the two left-most filters in Figure 3.

For this constraint, we used the term described in [7] for multiple concurrent elements:

$$L_{\text{bin}} = \sum_{p=1}^{M} \sum_{n=1}^{k} \sum_{j=1}^{d} \left| W_{pnj}^2 - W_{pnj} \right|$$

where $W \in \mathbb{R}^{M \times k \times d}$ is the weight tensor for the convolution layer, M is the number of filters (i.e. the number of patterns to learn), k is the number of action steps, and d is the number of features.

Note that this approach is not simply a hard thresholding of the weights (such as rounding) since that would be a non-differentiable operation—i.e. it would lack a well-defined derivative, creating complications for the back-propagation process used to train the model. While the absolute value function in our regularization term is also non-differentiable at exactly 0, it remains differentiable for every other case, allowing the model to learn via gradient descent.

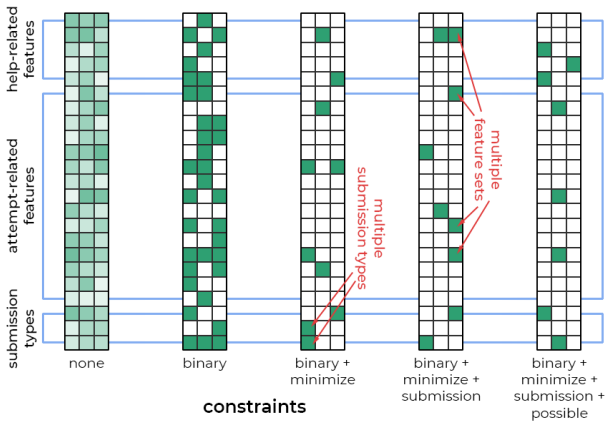

Figure 3: Visualization of the effects of increasing constraints on the model's convolutional filters.

3.1.2 *Minimize positive weights per action step*

An additional regularization term places a penalty on action steps that have a high number of positive weights. We introduced this term to encourage the model to focus only on the most relevant features per action step. We reasoned that having fewer activated features would also improve intelligibility by reducing cognitive load at explanation time.

The impact of this constraint can be seen in Figure 3: there are far fewer positive features in the third filter from the left compared to the second. Note that the weights in both of these example filters are already binarized.

To achieve this, we added the following term to the loss function:

$$L_{\text{min}} = \sum_{p=1}^{M} \sum_{n=1}^{k} \text{ReLU}(r^{(a - w_{pn})} - b)$$

where r controls the penalty rate, a sets the number of activated features at which the penalty begins to be applied, and b is a bias that further accelerates the penalty rate. These hyperparameters were set during tuning.

3.1.3 *Force single submission type*

In our dataset, every student action step has one submission type attached, whether a request for *help* in the form of a next-step hint, a *correct attempt*, or an *incorrect attempt*. Consequently, we introduced a regularization term to ensure that the model learns at most one submission type per action step.

Note that this is slightly simplified from the original features described in [17], which used two separate *incorrect attempt* labels: *wrong* and *bug*. Some of the expert patterns also use *attempt* to indicate any submission type that is not *help*. We removed this feature to simplify the submission types—all expert patterns that used it also included a non-help-related feature in the same action step, making it unnecessary.

The regularization term we used to enforce this constraint is:

$$L_{\text{sub}} = \text{ReLU}(s_{pn} - 1)$$

where s_{pn} refers to a squashed vector of a slice of W_{pnj} across dimension j—similar to the penalty term to minimize the number of positive weights as described above, but using only the features that correspond to submission types.

Figure 3 again shows the impact of this regularization term. The fourth filter from the left (where this constraint has been introduced) contains at most one positive submission type per action step (column), unlike those to its left which don't have this constraint.

3.1.4 Ensure possible feature combinations

Our final regularization term encourages the model to learn only positive weights in one of two feature sets for any single action step. For example, the nature of our feature set makes it so that the first five features are help-related features, so they can only be positive when the submission type is *help* (third feature from the bottom in the filters visualized in Figure 3). By extension, all other features (except submission types) are attempt-related features and can only be activated when the submission type is NOT *help*.

This means that the input data will never have a positive feature in both feature sets in the same action step. Filters that don't follow this guideline will never lead to a perfect match, so our constraint is designed to avoid such ineffective patterns. Compare the right-most filter in Figure 3 with the others to see the effect of this constraint.

The regularization term we used for this purpose is:

$$L_{\text{poss}} = \sum_{p=1}^{M} \sum_{n=1}^{k} \min\left(\left(\frac{\sum_{i \in S_h} W_{pni}}{\text{len}(S_h)}\right)^2, \left(\frac{\sum_{i \in S_a} W_{pni}}{\text{len}(S_a)}\right)^2\right)$$

where S_h and S_a are vectors containing the indices of help- and attempt-related features, respectively.

3.2 Explicit filter matching

Aside from the constraints implemented via regularization terms to the loss function, we introduced a set of final constraints on our model by altering the architecture of the model itself. In a typical convolutional neural network, the outputs of the convolutional layer, after passing through an activation function, become the inputs to a fully connected layer designed to turn them into a final probability prediction reminiscent of a logistic regression. As with the convolutional layer's weights, the weights of this fully connected layer are learned during training. Because these weights play an important role in shaping the model's final prediction, they would need to be included in the evidence used to create a fully faithful explanation.

In order to achieve our goal of allowing the convolutional filters to act as self-contained patterns of gaming-the-system behavior, and to ensure that a filter match unequivocally leads to a positive prediction and that the absence of a matching pattern leads to a negative prediction, we designed a novel fully connected layer architecture. Our layer diverges from the conventional linear transformation followed by a global activation function. Instead of this conventional approach, our layer employs an element-wise activation function—specifically, a scaled sigmoid that approximates a step function—applied immediately after the input is multiplied by the layer's weights and prior to the summation step.

The layer's inputs are the convolutional layer's outputs after passing through a max pooling layer that ensures only each filter's convolution with the most matching activations is passed on. The layer's weights (of length equal to the number of convolutional filters, M, and thus equal to the length of the layer's input filter), rather than being learned, are manually set to the inverse of the sum of all weights for each filter.

In essence, this makes it so that when the layer's inputs are multiplied element-wise by its weights, any filters that perfectly matched on the input will result in a 1, whereas all others will result in a 0. Note that a "perfect match" in this case exists when all positive weights in a filter (i.e. 1 after binarization) are also positive in the input instance. Negative weights in a filter (i.e. 0 after binarization) can have any value in the input instance.

This design allows the network to output a scalar value that decisively indicates a *match* (i.e., above or equal to 0.5) or *non-match* (i.e., below 0.5) based on the presence of any exact filter match. We refer to this mechanism as a *differentiable thresholding fully connected layer*, as it bridges the gap between hard thresholding (e.g. a conditional function) and standard fully connected layer soft aggregation, all while remaining trainable via gradient descent.

3.2.1 Formal description

Formally, we can describe the differentiable thresholding fully connected layer as follows.

Let the matrix $h \in \mathbb{R}^{M \times C}$ be the output of the convolutional layer (the feature maps) after passing through a ReLU activation function, where M is the number of filters in the convolutional layer and C is the number of convolutions. A max pooling function is applied to these feature maps to keep only the convolution with the most activations per filter. Therefore, let $f = \text{MaxPool}(h)$, where $f \in \mathbb{R}^M$.

This vector then serves as the input to the differentiable thresholding fully connected layer. Let the layer's weight vector be $w \in \mathbb{R}^M$. The thresholding layer processes the input in four distinct stages:

First, the weights w_p are manually set using the convolutional filters. If $W \in \mathbb{R}^{M \times k \times d}$ is the weight tensor for the convolutional layer, then for each filter p:

$$w_p = \frac{1}{\sum_{n=1}^{k} \sum_{j=1}^{d} \left|W_{p,n,j}\right| + \varepsilon}$$

where ε is a small constant added for numerical stability.

Second, each input element is multiplied by its corresponding weight:

$$z_i = f_i \cdot w_i, \quad \text{for } i = 1, \ldots, M$$

Third, a scaled sigmoid function is applied to z_i to approximate a step function:

$$a_i = \sigma\Big(t(z_i - \beta)\Big)$$

where $\sigma(\cdot)$ is the sigmoid function, t controls the steepness (with larger t making σ approach a step function), and β is an offset (typically set near 1, e.g. 0.99).

Fourth, a softmax function with a small temperature τ is used to compute weights that emphasize larger activations:

$$s_i = \frac{\exp\left(\frac{a_i}{\tau}\right)}{\sum_{j=1}^{n} \exp\left(\frac{a_j}{\tau}\right)}$$

Finally, the layer's final output is a weighted sum of the activated values:

$$y_{\text{thresholding}} = \sum_{i=1}^{M} a_i \, s_i$$

This formulation allows the layer to perform a nearly hard thresholding operation in a differentiable manner. The sigmoid approximates a step function, ensuring the network can decide between *match* (output ≥ 0.5) and *non-match* (output < 0.5) in a soft, gradient-friendly fashion.

3.2.2 Branching architecture

Notably, our architecture includes both our custom thresholding layer and a conventional fully connected layer, with the final output being a weighted sum of the two via a convex combination using the weight α. Like the γ_i scaling weights we use to modulate the regularization terms in our loss function, this branching approach allows us to control the impact of the thresholding layer. In this way, the model can have a more traditional warm-up period during training before our explicit filter matching via differentiable thresholding fully connected layer begins to influence the model's final predictions. Once this weight surpasses a certain (low) threshold, we also freeze the weights of the traditional fully connected layer so that the convolutional weights are the only parameters learned.

The traditional fully connected layer, or traditional branch of our architecture, processes the input through adaptive max pooling, flattening, a linear transformation, and a sigmoid activation:

$$y_{\text{trad}} = \sigma\Big(\mathbf{w}^\top \text{flatten}\Big(\text{pool}(x)\Big)\Big)$$

The model's final output is then computed as a weighted sum of the two branches:

$$y = (1 - \alpha)\, y_{\text{trad}} + \alpha\, y_{\text{thresholding}},$$

with an additional clipping to ensure that $y = \min\{y, 1\}$.

This approach serves as a differentiable mechanism that gradually shifts from a traditional fully connected layer to our custom thresholding layer as controlled by α.

3.3 Progressive constraining

In order to allow the model to properly learn over multiple epochs, we used the scaling weights γ_i to control the impact of each regularization term i independently, as well as α to control the impact of the differentiable thresholding fully connected layer. We allowed a warm-up period for each constraint, during which time its corresponding weight was gradually increased from 0 to its final value.

The warm-up periods were staggered, with each term having its own starting epoch and growth rate, which we treated as hyperparameters to be tuned. We used this staggered approach so that the model could begin incorporating a single constraint at a time, having greater flexibility at the beginning of the training process and progressively becoming more constrained, until finally converging on our desired filter format and inference approach.

We used grid search to find the optimal values for these hyperparameters, ensuring both that the model reached appropriate accuracy on the validation set and that the convolutional filters followed the constraints we set out to enforce. The grid search was set up with various possible starting epochs for each constraint, different growth rates, and different orders in which to introduce the constraints. Alongside accuracy on the validation set, we also exported visualizations of the learned filters at different points during training to compare.

We found that when the binarizing constraint was introduced too early, the weights had difficulty changing, even when the loss was high due to the penalties from other constraints. We also found that when the explicit filter matching constraint (via the differentiable thresholding fully connected layer) was introduced too late, the model often struggled to later learn patterns that matched the inputs exactly. Ultimately, we found that the best results were achieved when we introduced the constraints in the following order: explicit filter matching, possible feature combinations, single submission type, minimum positive weights per action step, and binary filters.

3.4 Training across multiple eras

Unfortunately, even with the gradual introduction of constraints, the model eventually reached a point where the gradients stopped flowing. This was likely due to the combination of the binary constraint and the explicit filter matching—along with its scaled sigmoid approximating a step function—which prevented the model from continuing to learn to the point of overfitting.

To address this potential underfitting, we introduced a mechanism to reset the training process at regular intervals. Extrapolating from the conventional use of the term *epoch*, borrowed from geologic chronology, we refer to these completed intervals of training epochs as training *eras*.

We trained our model for a total of 50 eras, each consisting of 200 epochs. At the end of each era, we reinitialized the weights for empty filters (those with all 0 weights) and for those with precision < 0.3. We also reset all γ_i values, restarting their staggered warm-up periods. This approach allowed the model to learn entirely new gaming-the-system patterns or refine existing ones by once again allowing gradients to

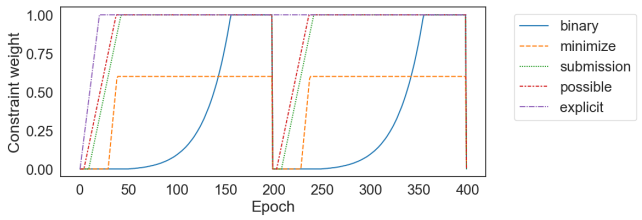

Figure 4: Scaling weight increases with staggered warm-up periods across two eras of 200 epochs each.

flow and giving the chance for the weights to escape local minima, all while still adhering to the constraints we set out to enforce. We did not reset the α scaling weight. Figure 4 shows the model's progressive constraining across two eras.

One advantage of our minimalist architecture—one with learnable weights only in the convolutional layer—is that we could save these weights to an external file at the end of each era, retaining the model's entire set of learned parameters for future use. This provided us a far larger set of gaming-the-system patterns than what the model could learn in a single era. By analyzing these patterns, we were able to identify the most effective ones, remove duplicates, and insert those remaining back into the model for final inference. We further describe this process and report our results in the next section.

4. RESULTS

Our approach generated a total of 102,400 filters (2,048 per era for 50 eras) from which we saved the 25,981 filters that achieved a precision above 0.3. Because the non-reinitialized filters were carried over to the initial state in subsequent eras, there was a large amount of repetition in the total set of filters identified during training. After removing duplicate filters, only 1,359 were unique. Furthermore, because of the nature of convolutional filters, if the positive weights in one filter are also positive in another, even if the latter has more positive weights, the former will match on the same inputs as the latter. In this scenario, the first filter has captured a more general gaming-the-system pattern than the second.

After accounting for these redundant filters, and keeping only the more general ones, we were left with 210 unique, non-redundant patterns.

From this set of patterns, we individually calculated each one's precision on our training set, which ranged from 0.302 to 1.0 with a mean of 0.610. We sorted the patterns by precision and calculated the cumulative Cohen's kappa on our validation set for each subset of n best patterns. The resulting metrics are shown in Figure 5.

Table 1: Performance metrics on various datasets.

Set	Accuracy	AUC	Kappa	Precision	Recall
train	0.940	0.923	0.541	0.499	0.672
val	0.917	0.883	0.380	0.360	0.513
test	0.909	0.847	0.319	0.324	0.422

Based on this analysis, we selected the top 132 patterns

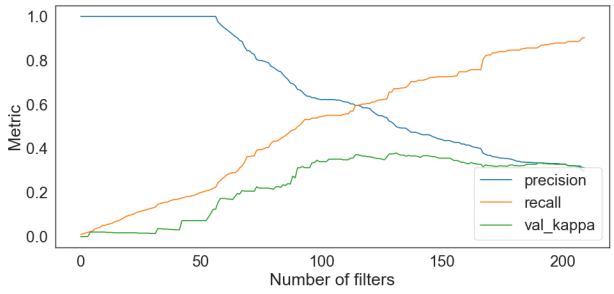

Figure 5: Cumulative metrics on patterns sorted descending by precision on the training set.

to use in our final model. We then evaluated the model's performance on our held-out test set, achieving a kappa of 0.319. This is slightly lower than the cognitive model's kappa of 0.330 on the same test set [17].

4.1 Comparison with expert patterns

We compared our model's 132 final learned filters with the expert patterns identified by [17]. However, of these 13 expert patterns, we had to make some small modifications to two of them to make them fully compatible (and comparable) with our filters. The two expert patterns in question contained an additional condition not reflected directly in the binary features: "*with at least one similar answer between steps*".

For example, one of these patterns was originally formulated as follows:

> ***help*** $\rightarrow$ ***incorrect*** $\rightarrow$ ***incorrect*** $\rightarrow$ ***incorrect***
> *(with at least one similar answer between steps)*

which we split into two variations that include the feature *similar answer* in key locations.:

> ***help*** $\rightarrow$ ***incorrect*** $\rightarrow$ ***incorrect*** *& [similar answer]* $\rightarrow$ ***incorrect***

> ***help*** $\rightarrow$ ***incorrect*** $\rightarrow$ ***incorrect*** $\rightarrow$ ***incorrect***
> *& [similar answer]*

This increased the number of expert patterns to compare up to 15.

Furthermore, because some expert patterns contained 2 or 4 action steps (as opposed to our filters' 3), we expanded them to all possible 3-action sequences for this analysis. From 2-action patterns we created two separate patterns: one with a blank action step at the beginning and another with a blank action step at the end. From 4-action patterns we also created two separate patterns: one encompassing the first three action steps and another the last three. While this latter scenario likely leads to trimmed patterns that the expert may no longer consider indicative of gaming-the-system behavior, we reasoned that such a comparison with our model's learned filters would still be informative.

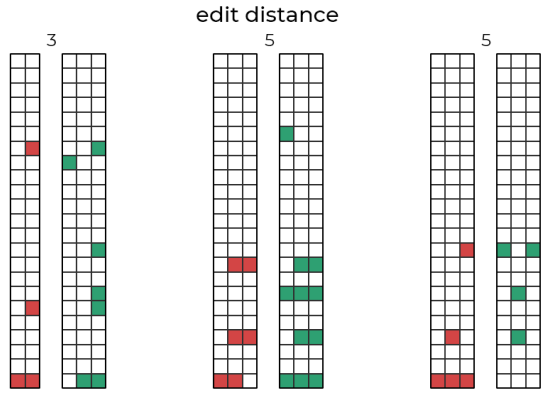

Figure 6: Expert patterns (red) and our model's learned filters (green) for the most similar pairs.

We made our comparison using the Levenshtein edit distance, which measures the minimum number of single-value edits (in our case, substitutions from 0 to 1 or vice versa) required to change one pattern into another. We found that the average distance between our filters and the expert patterns was 13.5, with a standard deviation of 2.6. This suggests that our model's learned filters differed quite significantly from those identified by human experts. Our model did not learn any of the expert patterns on its own. The most similar patterns had an edit distance of 3 and 5, but only three pairs were this similar (shown in Figure 6).

One noticeable difference is that our model's learned filters tended to have many more positive features than the expert patterns. The model's filters had a mean of 12.0 positive features, with a range of 4 to 18 and a standard deviation of 2.3. This is in contrast to the expert patterns, which—when taking into account only those with three action steps—had a mean of 4.9 positive features, with a range of 4 to 7 and a standard deviation of 1.1. The entire set of expert patterns had a mean of 5.1 positive features, with a range of 4 to 7 and a standard deviation of 1.0.

We also measured the differences between the model's own learned filters pairwise. The mean edit distance between two filters was 12.9, with a range of 2 to 26 and a standard deviation of 3.7. The smallest edit distance between two filters was 2, which occurred 13 times, while the next smallest edit distance was 3, which occurred 25 times.

5. DISCUSSION

Our model achieved performance comparable with the cognitive model described in [17], with a slightly lower kappa on the held-out test set. Its precision was higher than the cognitive model's (0.324 vs. 0.307), while its recall was lower (0.422 vs. 0.528). It outperformed the machine learning model created by [2]. This suggests that our model was able to learn patterns indicative of gaming-the-system behavior.

Surprisingly, despite the many eras during which our model was trained and reinitialized with new starting weights, and despite the high number of convolutional filters (2,048), the final number of usable filters was only 210. Only 5.2% of our total number of filters with precision > 0.3 were unique. The model therefore repeatedly learned many of the same filters, even while it had the opportunity to learn many new ones.

One possibility is that the constraints we introduced were too restrictive, causing the model to converge repeatedly on local minima. This is supported by the fact that the model's filters had more positive features than the expert patterns on average—the very outcome that our $L_{\min}$ regularization term tried to prevent.

Our more complex filters (i.e. with more positive features) help explain the model's higher precision but lower recall compared to the cognitive model. The model's filters were more specific, leading to fewer false positives, but they were also less encompassing, leading to more false negatives. This also explains the two models' similar overall performance despite our model having many times more patterns than the cognitive model.

Despite this, the final set of learned filters do successfully predict gaming-the-system behavior when compared to the cognitive model with similarly structured patterns. While our model uses many more patterns than the cognitive model, its performance on the held-out test set indicates that it did not overfit to the training data. Given this, the question remains of how well these filters can be used to create effective explanations.

5.1 Explainability

We set out to create a model that was fully interpretable, meaning that we could extract sufficient evidence from it to create an explanation that (1) is faithful, fully capturing the model's inference process, while (2) remaining intelligible.

By ensuring that the model's only learnable parameters are contained in the convolutional filters, and by constraining those filters to follow the template of patterns created by human experts, we believe we have achieved our goals. The weights of the convolutional layer provide 100% explanatory potential [23]. They can be clearly interpreted as sequential patterns that emulate the binary presence or absence of specific student actions.

These weights, paired with this interpretation, can thus be used to create different kinds of explanations. For example, if the model detects gaming-the-system behavior in a student's actions because they matched a specific filter, that filter's weights can be used to explain why the model made that decision. These explanations can take many forms, such as visual matrices for more technical end-users, simplified bullet-point explanations, or even text-based explanations in user-friendly language using LLMs (see the examples in Figure 7).

This is an example of local explainability, where a specific prediction is explained. More global patterns can also be extracted from the model's filters, such as the most common patterns or the most important features. These can be used to create more general explanations, such as identifying the most common reasons a student is flagged for gaming-the-system behavior or the most important features to look for. Through all of this, the model's filters provide a direct link to the entirety of the model's learned parameters, allowing

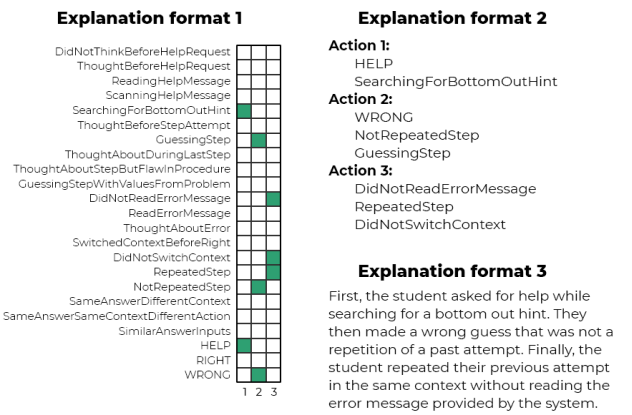

Figure 7: Example explanations created from the model's learned filters.

for fully faithful explanations.

The implications of this are significant. If such fully interpretable models are possible and can retain sufficient flexibility to fit a wide range of use cases, they could be used to create more transparent AI systems in education. Our contribution is not the specific model we have trained, but rather the general constraints-based approach we have outlined, along with the evidence we have provided that it functions as intended.

However, before making strong claims about interpretability, it is clear that we need to evaluate the explanations our approach can generate. Fortunately, the explanations we have proposed here are the perfect test case for the human-grounded evaluation methods outlined by [5]. There is also room for future work to further refine our approach.

5.2 Limitations and future work

While our approach so far indicates the successful learning of interpretable patterns for behavior detection, there remain important limitations to address. The most pressing of these is the issue we encountered with the model's gradients no longer flowing after a certain number of training epochs. Experiments we conducted with disabling individual constraints while tracking gradient norms indicated that the issue arises from the combination of the binary constraint and the explicit filter matching via thresholding layer.

We managed to produce a workable model with functioning filters through the use of multiple training eras, but this is not an ideal solution. There may be specific transformations where the gradients are being blocked that could easily be addressed. For example, it may simply be that the thresholding layer's scaled sigmoid function becomes too steep too quickly. Most likely, the issue involves multiple transformations interacting in a way that is difficult to predict. Broadening our hyperparameter search to include such variables would be one possible solution, though it is not clear how effective this brute force approach would be. We plan to systematically investigate this issue further in future work.

One limitation of our overall approach to explainability is that it relies on carefully engineered interpretable features. We were able to rely on those previously crafted by gaming-the-system experts using cognitive task analysis [17], but in many real-world scenarios, such features may not be available.

Another, less insurmountable, limitation of the current model is that the convolutional filters only allow for a fixed number of action steps. That is, the input may be of any length, but the patterns can only have at most three student action steps. This is an inherent limitation of the architecture, since the tensor containing the convolutional weights forces a fixed kernel size. One possible solution to explore in future work would be to provide separate but parallel convolutional layers, one for each desired sequence length. The feature maps, or output of each layer, could then be concatenated and passed through the thresholding layer as normal.

As we have mentioned, future work should seek to evaluate the explainability of the explanations derived from our convolutional filters. Following [5], we plan to conduct two human-grounded evaluation experiments: forward simulation and counterfactual simulation. By asking participants to predict the model's output and to identify changes to inputs that will lead to different outputs, we can measure the extent to which our explanations are intelligible and faithful.

Finally, we plan to conduct an ablation study to measure the impact of each constraint on the model's accuracy and on the filters' interpretability. This will help us better understand the interplay between constraints and between interpretability and accuracy. It may also provide insights into the issue of gradients no longer flowing.

6. CONCLUSION

We have described a novel approach to creating a neural-network-based behavior detection model that is interpretable by design. By constraining the model's learning process through a series of regularization terms and architectural changes, we were able to create a model that learned patterns indicative of gaming-the-system behavior. These patterns emulate the structure of those identified by human experts, indirectly indicating that their interpretations are sound.

Importantly, the parameters pertaining to these patterns encompass all the learnable weights in the final model, providing 100% explanatory potential to the model's inner workings. We demonstrated some possible ways in which these patterns can be used to create explanations for different potential audiences, all while remaining fully interpretable.

We have not yet conducted a systematic evaluation of these explanations, but we believe they have the potential to be both faithful and intelligible to human end-users. We indicated some promising ways to evaluate these claims via forward simulation and counterfactual simulation tasks as human-grounded evaluation experiments. We will conduct this evaluation in future work, along with an investigation into the model's gradient issues and an ablation study to better understand the interplay between flexibility and interpretability.

7. ACKNOWLEDGMENTS

This study is supported by the National Science Foundation under Award #1942962. Any conclusions expressed in this material do not necessarily reflect the views of the NSF.

8. REFERENCES

[1] D. Alvarez Melis and T. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[2] R. S. Baker and A. M. J. A. de Carvalho. Labeling student behavior faster and more precisely with text replays. In *Proceedings of the 1st International Conference on Educational Data Mining (EDM)*, pages 38–47, 2008.

[3] N. Bosch and L. Paquette. Metrics for discrete student models: Chance levels, comparisons, and use cases. *Journal of Learning Analytics*, 5(2):86–104, 2018.

[4] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8), July 2019.

[5] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning, Mar. 2017.

[6] A. Jacovi and Y. Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4198–4205, Online, July 2020. Association for Computational Linguistics.

[7] L. Jiang and N. Bosch. Predictive sequential pattern mining via interpretable convolutional neural networks. In *Proceedings of the 14th International Conference on Educational Data Mining (EDM 2021)*, pages 761–766, 2021.

[8] H. Khosravi, S. B. Shum, G. Chen, C. Conati, Y.-S. Tsai, J. Kay, S. Knight, R. Martinez-Maldonado, S. Sadiq, and D. Gašević. Explainable artificial intelligence in education. *Computers and Education: Artificial Intelligence*, 3:100074, Jan. 2022.

[9] S. Krishna, T. Han, A. Gu, J. Pombra, S. Jabbari, S. Wu, and H. Lakkaraju. The disagreement problem in explainable machine learning: A practitioner's perspective, Feb. 2022.

[10] D. Kumar and M. A. Mehta. An overview of explainable AI methods, forms and frameworks. In M. Mehta, V. Palade, and I. Chatterjee, editors, *Explainable AI: Foundations, Methodologies and Applications*, volume 232, pages 43–59. Springer International Publishing, Cham, 2023.

[11] Q. Liu, J. D. Pinto, and L. Paquette. Applications of explainable AI (XAI) in education. In D. Kourkoulou, A.-O. O. Tzirides, B. Cope, and M. Kalantzis, editors, *Trust and Inclusion in AI-Mediated Education: Where Human Learning Meets Learning Machines*, pages 93–109. Springer Nature Switzerland, Cham, 2024.

[12] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] A.-p. Nguyen and M. R. Martínez. On quantitative aspects of model interpretability, July 2020.

[14] L. Paquette and R. S. Baker. Variations of Gaming Behaviors Across Populations of Students and Across Learning Environments. In E. André, R. Baker, X. Hu, M. M. T. Rodrigo, and B. Du Boulay, editors, *Artificial Intelligence in Education*, volume 10331, pages 274–286. Springer International Publishing, Cham, 2017.

[15] L. Paquette and R. S. Baker. Comparing machine learning to knowledge engineering for student behavior modeling: A case study in gaming the system. *Interactive Learning Environments*, 27(5-6):585–597, Aug. 2019.

[16] L. Paquette, R. S. Baker, A. de Carvalho, and J. Ocumpaugh. Cross-System Transfer of Machine Learned and Knowledge Engineered Models of Gaming the System. In F. Ricci, K. Bontcheva, O. Conlan, and S. Lawless, editors, *User Modeling, Adaptation and Personalization*, volume 9146, pages 183–194. Springer International Publishing, Cham, 2015.

[17] L. Paquette, A. M. de Carvalho, and R. S. Baker. Towards understanding expert coding of student disengagement in online learning. In *Proceedings of the 36th Annual Cognitive Science Conference*, pages 1126–1131, 2014.

[18] R. Pelanek. Metrics for evaluation of student models. *Journal of Educational Data Mining*, 7(2), 2015.

[19] J. D. Pinto and L. Paquette. Deep learning for educational data science. In D. Kourkoulou, A.-O. Tzirides, B. Cope, and M. Kalantzis, editors, *Trust and Inclusion in AI-Mediated Education*, pages 111–139. Springer Nature Switzerland, Cham, 2024.

[20] J. D. Pinto and L. Paquette. Towards a unified framework for evaluating explanations. In *Joint Proceedings of the Human-Centric eXplainable AI in Education and the Leveraging Large Language Models for Next Generation Educational Technologies Workshops (HEXED-L3MNGET 2024)*, volume 3840, Atlanta, Georgia, USA, 2024. CEUR-WS.

[21] J. D. Pinto, L. Paquette, V. Swamy, T. Käser, Q. Liu, and L. Cohausz. Preface to the proceedings of the Human-Centric eXplainable AI in Education Workshop (HEXED 2024). In *Joint Proceedings of the Human-Centric eXplainable AI in Education and the Leveraging Large Language Models for Next Generation Educational Technologies Workshops (HEXED-L3MNGET 2024)*, volume 3840, Atlanta, Georgia, USA, 2024. CEUR-WS.

[22] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier, Feb. 2016.

[23] M. Rizzo, A. Veneri, A. Albarelli, C. Lucchese, M. Nobile, and C. Conati. A theoretical framework for AI models explainability with application in biomedicine. In *2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–9, Eindhoven, Netherlands, Aug. 2023. IEEE.

[24] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[25] H. Scheers and T. De Laet. Interactive and explainable advising dashboard opens the black box of student success prediction. In T. De Laet, R. Klemke, C. Alario-Hoyos, I. Hilliger, and A. Ortega-Arranz,

editors, *Technology-Enhanced Learning for a Free, Safe, and Sustainable World*, volume 12884, pages 52–66, Cham, 2021. Springer International Publishing.

[26] H. Suresh, S. R. Gomez, K. K. Nam, and A. Satyanarayan. Beyond expertise and roles: A framework to characterize the stakeholders of interpretable machine learning and their needs. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–16, New York, NY, USA, May 2021. Association for Computing Machinery.

[27] V. Swamy, S. Du, M. Marras, and T. Kaser. Trusting the explainers: Teacher validation of explainable artificial intelligence for course design. In *LAK23: 13th International Learning Analytics and Knowledge Conference*, pages 345–356, Arlington TX USA, Mar. 2023. ACM.

[28] V. Swamy, J. Frej, and T. Käser. The future of human-centric eXplainable Artificial Intelligence (XAI) is not post-hoc explanations, July 2023.

[29] M. Tissenbaum and J. Slotta. Supporting classroom orchestration with real-time feedback: A role for teacher dashboards and real-time agents. *International Journal of Computer-Supported Collaborative Learning*, 14(3):325–351, Sept. 2019.