Data Synthesis and Simulation for Modeling Cognitive Abilities

Gergely Füstös and Bertalan Forstner

Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3, H-1111 Budapest, Hungary
fustosgergely@edu.bme.hu, bertalan.forstner@vik.bme.hu

Abstract: Recent advancements in the methodology of cognitive assessment and development rely on various cognitive models. Determining the underlying abilities tapped by individual tasks involves different procedures, which presuppose dependencies on specific subskills, considerations of statistical distributions, and a substantial amount of measurement data for accurate estimation of latent factors. Addressing these bottlenecks, various deep learning (DL) models show promising performance. Despite their initial success, it is evident that DL models are hindered by the requirement for significant quantities of annotated and labeled data to experiment and refine these models. Synthetic data offer a solution to this challenge by being easily generated, error-free, inexhaustible, pre-annotated, and circumventing various ethical and practical concerns. The past decade has witnessed remarkable progress in data synthesis and domain adaptation techniques, narrowing the statistical gap between synthetic and real data. Beyond sustaining the DL revolution, synthetic data will pave the way for the next generation of DL models, capable of understanding the physical composition of the world and learning continually, multimodally, and interactively. This paper clarifies the models emerging from prevalent cognitive models, statistical methodologies, and psychometric research regarding the subjects and their subskills, as well as how to model the parameter and subskill dependencies of individual tasks independent of the limitations posed by current solutions. Building upon these insights, an environment for data synthesis and simulation is developed, suitable for validating various analysis solutions.

Keywords: cognitive modeling; data synthesis; simulation

1 Introduction

In recent years, significant efforts have been made to enhance the abilities of both typical and atypical learners, particularly children with learning disorders, through the more efficient development of various cognitive and other tasks. The difficulty levels of individual developmental tasks are set using various approaches, aiming to prevent tasks from being either too easy (leading to boredom) or too difficult

(resulting in task abandonment). The most widely accepted procedures are based on positive psychology, attempting to set the difficulty of challenges so that individuals can solve them with a probability of 60-80%. A successfully calibrated series of challenges can keep the individual in a state of flow.

The goal of the paper is to develop a simulation environment that enables the synthesis of cognitive ability datasets by generating various types of examinees and parametrized tasks. This environment is designed to facilitate research on cognitive modeling, allowing researchers to validate analytical approaches and test machine learning-based ability estimation methods without the constraints of real-world data scarcity.

Our motivation for the research stems from the limitations of current cognitive assessment methodologies, which require vast amounts of empirical data for accurate estimation of latent cognitive traits. Many existing approaches, particularly those relying on deep learning, struggle due to the need for extensive annotated training data. Synthetic data offers a promising alternative, as it is easily generated, error-free, and pre-annotated, allowing for greater flexibility in experimental design and validation.

Recent advancements in the field of cognitive science have seen a notable evolution in formal intelligence models since Spearman's groundbreaking revelation regarding the positive manifold [7], which signifies the consistent occurrence of positive correlations among cognitive test scores. Contemporary approaches have departed from conventional reflective latent factor models to introduce diverse emergence mechanisms. Among these, prominent models include sampling models [8], gene-environment interaction models [9], and network models [10]. The amalgamation of multifactor and hierarchical theories has notably culminated in the well-endorsed Cattell-Horn-Carrol (CHC) theory [11], which encompasses fluid and crystallized intelligence alongside a three-tier hierarchy.

The challenges are multifaceted. First, cognitive developmental tasks are inherently complex, as they often require multiple cognitive subskills to be performed simultaneously. For example, a puzzle game may require visual processing, short-term memory, and logical reasoning, making it difficult to isolate the impact of individual sub-abilities. Current research focuses on developing assessments that can measure sub-abilities independently. Various studies are underway to improve such assessments [12], including the Wechsler Adult Intelligence Scale—Fourth Edition (WAIS–IV) [6], which aims to evaluate cognitive components such as crystallized ability, fluid reasoning, visual processing, short-term memory, processing speed, and quantitative reasoning. However, obtaining accurate estimations of these sub-abilities remains an open problem. The second major challenge is the parameterization of task complexity. The relationship between task difficulty and cognitive demand is often nonlinear; for example, increasing the size of a puzzle board from 6x6 to 7x8 may disproportionately affect difficulty.

Moreover, machine learning-based approaches require substantial labelled datasets, and selecting relevant features in high-dimensional spaces remains a critical issue.

This paper introduces a novel approach by the development of a simulation environment capable of addressing these challenges by generating structured, parametrized synthetic data. This approach allows for flexible experimentation, enabling researchers to analyze the interaction between cognitive subskills and task success, validate cognitive models, and benchmark different machine learning approaches for ability estimation. Unlike conventional approaches that depend on large-scale empirical data collection, this framework provides a reproducible and controlled setting for hypothesis testing and model evaluation.

Our work thus contributes to the universal interpretability and reusability of cognitive ability datasets, facilitating the acquisition and sharing of training and testing datasets within the machine learning research community, enabling the comparison of results from different machine learning tools, and accelerating the overall research process.

The paper is organized as follows. The next section presents the relevant elements of the research area and the existing simulation environments. The third section analyses the model and background of the simulator. The fourth section details the software aspects of the work, followed by the summary of the paper to conclude.

2 Related Work

The CHC model widely defines individuals' abilities. While most item-response or task-solving tests assume a single latent ability, the probability of solving skill-building games and similar tests often relies on multiple abilities in different ways. One identified bottleneck in research is the lack of a solution that can estimate the probability of task solving for a subject with known abilities when dealing with tasks based on multiple dimensions (that is, partial or sub-abilities). The Multidimensional Item Response Theory (MIRT) [1] is a statistical modeling framework used for analyzing multidimensional tests and modeling item responses. For the MIRT model to be appropriately applied, several important assumptions need to be considered, of which three are highlighted: (1) MIRT models often require larger sample sizes compared to unidimensional IRT models to estimate parameters accurately, especially when dealing with a high number of dimensions or items. (2) Normality of Latent Variables: The MIRT model assumes that latent variables (abilities) follow a normal distribution. This means that the distribution of abilities among the test takers is normal in the population. (3) Assumption of Local Independence: MIRT assumes local independence, meaning that conditional on the latent traits, item responses are independent. Violations of this assumption can lead to biased parameter estimates and model misfit.

Additional extensions of the model have been proposed. For instance, [2] introduces a novel model known as the Multidimensional Two-Parameter Logistic Model with Ability-Item-Based Guessing (M2PL-AIG) model. This model not only reduces the number of parameters compared to the M3PL model but also incorporates participants' guesses, similar to the M3PL model.

Computerized adaptive testing (CAT) represents a sophisticated method for administering assessments, surveys, and similar evaluations. It relies on intricate computer algorithms that tailor the test to each individual examinee, while also managing practical considerations such as content distribution, item exposure, and test duration. CATs have demonstrated the capability to significantly decrease test length by as much as 90% without sacrificing precision. Nevertheless, realizing such efficiency gains and leveraging the benefits of CAT necessitates thorough research studies conducted by CAT developers to simulate CAT performance. [3]

For achieving these goals, several popular simulators are available. Among them, CATSim [4] offers three simulation types for computerized adaptive testing (CAT), employing both dichotomous and polytomous item response theory (IRT) models: post-hoc (real data) simulations, hybrid simulations, and Monte Carlo simulations. CATSim enables simulations for item banks containing up to 999 items, with no restriction on the number of examinees for both post-hoc and hybrid simulations, and a cap of 10,000 examinees for Monte Carlo simulations.

Similar goals are pursued by SimulCAT [5], a software program developed for conducting CAT simulation studies. The SimulCAT program encompasses most of the methods utilized for the three key components of item selection in CAT: item selection criterion, item exposure control, and content balancing. It also includes methods for determining test length (fixed or variable) and score estimation algorithms.

However, a strong criticism against these simulators is that they are not suitable for generating examinees with multiple different sub-abilities, and it is not possible to configure the distribution of these subskills. The mentioned simulators can only apply a single parameter with a normal distribution in terms of ability.

In the analysis of the aforementioned and related research works, the following challenges emerged:

(A) Data Synthesis

To develop and validate ability-enhancing tasks, simulations are needed where (1) the subskills of subjects can be defined within flexible frameworks, and (2) the tasks to be tested can be parameterized, where the likelihood of solving them depends on reaching certain levels of subskills, and the dependency on these can be weighted (allowing for the simulation of well-known compensatory abilities in psychology).

(B) Simulation

A simulation environment is needed that can test various cognitive models. In such cases, the simulator is used to validate how accurately a procedure can replicate latent traits by observing the success of solving certain types of tasks. This is particularly important in assessing the success of neural network-based solutions developed in response to deficiencies and challenges in solutions based on factor analysis and decision trees.

3 The Simulator

The cognitive simulator is constructed using small, interlocking foundational blocks, enabling it to perform multiple simulation tasks. These tasks include simulating an individual's response to a parameterized exercise based on the given parameters and the person's abilities, as well as simulating numerous artificial entities and exercises and their corresponding responses. In the subsequent sections, we will explore the functionality of each of these foundational blocks, beginning with the core process: calculating the probability of a correct answer for a known individual undertaking a specified task.

3.1 Predicting Exercise Outcome

The simulator's core component processes the attributes of both a person and an exercise, computing the likelihood that the individual will correctly solve the specific task. It can provide the outcome as either a probability or a binary value.

Throughout the article, we represent the person's abilities with a single vector, and the exercise parameters with one mandatory and another optional vector. These vectors correspond to specific cognitive abilities; therefore, prior to using the simulator, the user must define all the specific sub-abilities they wish to consider. For instance, one could utilize the ten broad cognitive abilities for this purpose.

For this study, we define a constant list of specific cognitive abilities, which will be referenced throughout the article to ensure consistent analysis and discussion. We denote this list as $abilities = ["Gv","Gq",\dots,"Ga"]$, where each element corresponds to a broad factor of cognitive processing as described by the CHC theory. For instance, *Gv* represents *Visual Processing*, *Gq* stands for Qua*ntitative Knowledge*, and *Ga* denotes *Auditory Processing*. The actual definitions of these abilities are not the primary focus; instead, their inclusion serves to facilitate easy reference. Thus, when $abilities_i$ is mentioned, it refers to one of these predefined sub-abilities in our list.

Finally, before detailing the attributes of the exercise-solving core component, let k represent the length of the previously defined $abilities$ list. This value denotes the number of specific sub-abilities we have chosen to include in our model.

3.1.1 Person Attributes

Let $\underline{\theta}$ represent the abilities of a person, where:

- $\underline{\theta} \in \mathbb{R}^k$
- $\forall i, 1 \leq i \leq k: \theta_i \sim \mathcal{N}(1, 0.15)$

Here, θ_i corresponds to the person's proficiency in the i th specific cognitive ability as outlined in the list $abilities$. This statistical representation allows us to model each cognitive ability as an independent quantifiable trait, broadening the conventional statistical approach where cognitive ability as a whole is typically represented by a single value following a similar normal distribution.

3.1.2 Exercise Attributes

Let $\underline{\beta}$ represent the difficulty of an exercise, and use $\underline{w}$ for a weighting purpose, where:

- $\underline{\beta} \in \mathbb{R}^k$
- $\underline{w} \in \mathbb{R}^k$
- $\forall i, 1 \leq i \leq k: \beta_i \sim \mathcal{N}(1, 0.15)$
- $\forall i, 1 \leq i \leq k: w_i \geq 0$

Here, β_i refers to the minimum level of proficiency required in the i th cognitive ability to solve the task correctly. By definition, the probability that a person with $\delta_i = r$ successfully solves an exercise requiring $\beta_i = r$ is almost 1, provided that only the i th cognitive ability is considered. On the other hand, if the person's ability is approximately two standard deviations (0.3) lower than β_i, then the probability is about 0. If the task does not involve any knowledge from one cognitive ability, the corresponding β_i should be set to zero.

The optional weight vector's i th element, w_i, defines the significance of the person's ability meeting the required level, β_i. If the weight vector is not provided, the importance of each specific ability is assumed to be equal. If $\beta_i > 0$, the weight also must be positive: $w_i > 0$. On the other hand, if $\beta_i = 0$, the weight could still be positive, because it gets automatically neglected by the simulator. This architecture was chosen, so one can use a vector containing only e.g., 1-s and does not have to manually set all weights to zero, where $\beta_i = 0$.

Prior to computation, the weight vector is normalized such that a value of 1 indicates normal importance, whereas values lower or higher denote lesser or greater

importance, respectively. Thus, the normalization considers the number of abilities required to solve a task, which is the number of positive values in vector $\underline{\beta}$. Let l denote this value. The normalization formula is as follows:

$$w_i' = \frac{w_i}{\sum_{i=1}^{k} w_i} \cdot l$$

This calculation ensures, that if all weights are equal, then the originally not zero values will all be 1-s, by normalizing the weights between 0 and 1, and then scaling them so they sum up to l – the number of important abilities.

We chose to separate the exercise attributes into difficulty and weight vectors because the roles they represent are distinctly different. Imagine a scenario, where an individual is required to solve equations presented on a screen. In this case, a basic lever of visual knowledge is crucial: the person must accurately perceive every detail of the equation. However, a higher proficiency in visual knowledge is not necessary to solve this task correctly. With our method we can accurately represent these types of cases by assigning a low value to the difficulty and increasing the importance.

Now that the attributes are defined, let's examine, how the simulator calculates the probability of an individual solving a specific exercise – given both the person's and the task's attributes.

The basis of the calculation lies in a statistical formula commonly used in psychometrics, known as Item response theory (IRT). The theory defines the item response function, which determines the probability of a correct response to one dichotomous item, considering the proficiency of a person and the difficulty of a task. The simulator employs a modification of the two-parameter IRT model to calculate probability with respect to one specific cognitive ability. The modified function is as follows.

$$p(correct \mid \theta_i, \beta_i) = \frac{1}{1 + e^{-d(\theta_i - (\beta_i - \sigma^2))}}$$

In the previous equation, d and σ^2 are constants, with the latter representing the standard deviation previously set as 0.15. The former d controls maximum steepness of the function and has been set to 20 to maintain the relationship between the attributes as defined. With this formula, only considering one ability, the probability of correct response is 0.5 if $\theta_i = \beta_i - \sigma^2$, around 1 if $\theta_i \geq \beta_i$, and around 0 if $\theta_i \leq \beta_i - \sigma^2$.

To calculate the probability considering all specific cognitive abilities we have employed the assumption, that the events of a person solving an exercise given one ability are independent, allowing the probabilities to be multiplied. We also accounted for the exercise weights and some unknown, random factors, leading us to the formula presented in the next.

$$p(correct \mid \theta, \beta, w) = \prod_{i=1}^{k} p(correct \mid \theta_i, \beta_i + x_i)^{w'}$$

$$x_i \sim \mathcal{U}_{[-\epsilon,\epsilon]}$$

As the formula indicates, the normalized weights are used as exponents, enhancing importance when the current weight exceeds 1, and reducing it below 1. Additionally, a random value between $-\epsilon$ and ϵ is added to each proficiency value to account for variables such as luck, lack of attention, and other similar factors. In cases where we are solely interested in whether the solution is correct, a calculated probability of 0.5 or higher will be considered a correct solution. Thus, we have arrived to the foundational formula of the simulator.

3.2 Simulating Results for Simple Exercises

In this section we will demonstrate how the derived formula can be applied to simulate the solving of simple exercises. By simple, we refer to tasks for which the vectors $\underline{\beta}$ and $\underline{\theta}$ are predetermined and independent of other task attributes, such as the difficulty level. We will focus on these more complex tasks in the subsequent section.

The simulator presents this feature as the form of the *simulate exercise* function. The function's input parameters are as followed:

- $n \in \mathbb{Z}^+$, the number of solutions to simulate
- $\underline{\beta}, \underline{w} \in \mathbb{R}^k$, vectors of the exercise
- $binary \in \{True, False\}$, whether to simulate probability of correct answer or merely classify the outcome as solved or not solved.

The function generates n abilities vectors representing individuals and calculates the probability of a correct response to the specified exercise. If the $binary$ parameter is set to $True$, the simulator will output binary values based on these probabilities. The generated abilities vectors are stored in a matrix, where each row corresponds to a person's abilities. The function returns this matrix alongside the results, stored in a separate vector.

3.3 Simulating Results for Parameterized Exercises

In real-life exercises, particularly in games, there are often multiple levels of tasks to undertake. The previous approach necessitates a separate exercise vector for each task variation, which can be both tedious and monotonous to manually calculate and configure. In this section, we explore how the simulator can simplify this process for users.

We can define higher-level parameters that are adjustable within a predefined range of values and control the exercise's exact $\underline{\beta}$ vector. For instance, consider a task where individuals must compute the results of various arithmetic operations. One parameter could specify the maximum number digits between operands. This illustrates the complex relationship between a parameter and the task's difficulty. Another parameter could specify the allotted time for an exercise. In this example, a larger value for this parameter corresponds to a less difficult task.

We have defined three types of higher-level parameters based on the relationship between the parameter's value and the task difficulty: linear, power, and exponential. However, these can be easily expanded to include new ones. First, we will explore the common features shared by these types.

Each parameter type must specify a range from which the parameter can take values. By default, the minimum value is zero and the maximum is ten, which is a common choice when defining levels. The value of every parameter can be set to any real number within this specified range, making the parameters stepless.

Each parameter must define a specific sub-ability it influences. This can be achieved by providing the index of the corresponding ability from the globally defined $abilities$ list. If a parameter affects multiple abilities, the relationship between the parameter each ability often varies. Therefore, we encourage users to create multiple separate variables for such scenarios.

Lastly, every parameter must have a maximum change value, which defines the difficulty multiplier for the largest parameter value. This value can be any positive number; values above 1 indicate a positive correlation with difficulty, while values below 1 indicate a negative correlation. For reference, the difficulty multiplier for the smallest value is set at 1. The parameters differ in how they transform the difficulty multiplier from 1 to the desired maximum change value.

Along with precise definitions of the previous values, a parameterized exercise also requires a base difficulty vector and a weight vector. A task can incorporate multiple parameters, each multiplying the difficulty value of the chosen abilities based on the parameter's current value.

Before we examine each type of parameter, let's denote the parameter's current value as x, the parameter's minimum and maximum as $\min(x)$ and $\max(x)$, respectively. We will denote the difficulty multiplier for the largest parameter as m_{max}, and the difficulty multiplier at a given parameter value as $m(x)$. Additionally, define the normalized value of x as $x_{normalized} = \frac{x - \min(x)}{\max(x) - \min(x)}$. This normalization scales x from its original range to a normalized range $[0, 1]$.

3.3.1 Linear Parameters

The simplest form of the previously described higher-level parameters are linear parameters. These change the difficulty multiplier linearly from 1 to the desired value when the parameter is adjusted from its minimum to its maximum. This functionality is achieved by the following function, computing the current multiplier for a given parameter.

$$m_{linear}(x) = 1 + (m_{max} - 1) \cdot x_{normalized} \quad (1)$$

Equation (1) defines a linear function ranging from 1 to m_{max}, when the parameter is between its minimum and maximum.

3.3.2 Power Parameters

The second type of higher-level parameters are power parameters. These change the difficulty multiplier using a power function such as x^l, where l is the exponent that can be defined as desired. The formula for this type of parameter is as follows.

$$m_{power}(x) = 1 + (m_{max} - 1) \cdot x_{normalized}^{l} \quad (2)$$

Because $x_{normalized}$ ranges between 0 and 1, $x_{normalized}^{l}$ will also be within this range. Therefore, the function $m_{power}(x)$ represents a power function that ranges from 1 to m_{max}, as the parameter varies within its range.

3.3.3 Exponential Parameters

Lastly, there are exponential parameters, which change the difficulty multiplier using parameter's value as an exponent.

$$m_{exp}(\mathrm{x}) = m_{max}^{x_{normalized}} \quad (3)$$

Equation (3) defines an exponential function, where the output ranges from 1 to m_{max}, as the parameter runs from its minimum to its maximum.

The simulator features a function called *simulate_exercise_params* designed to generate solutions for parameterized tasks. This function extends the features of a simple exercise simulation, and has two additional inputs:

- *List of parameter types:*

  This specifies the type and settings of each parameter involved in the exercise. For example:

  - A linear parameter affecting $abilities_2$, with a minimum value of 0, a maximum of 5, and a maximum multiplier being 2.

- *List of current parameter values:*

This provides the current values for each parameter at the time of the exercise simulation. The values should correspond to the types defined in the list of parameter types.

Like the simple exercise simulation, this function generates n abilities vectors representing individuals and calculates the probability of a correct response based on the current parameter values. Additionally, it can also output binary values. The abilities vectors are stored in matrix similarly and are returned along with the results, which are contained in a separate vector.

3.4 Data Synthesis

The previously defined simulator also serves as data synthesis. It employs widely accepted and empirically verified predictive formulas from the field of study, enabling it to generate synthesized data as well.

The simulator presents a *data_synthesis* function, for which the input is only 3 number, and a binary value:

- $n \in \mathbb{Z}^+$, the number of individuals
- $m \in \mathbb{Z}^+$, the number of exercises
- $k \in \mathbb{Z}^+$, the number of specific sub-abilities, therefore the length of the $abilities$ list
- $binary \in \{True, False\}$, similarly, as described earlier

The data synthesis function generates n abilities vectors, each sub-ability defined by a normal distribution centered around 1 with a standard deviation of 0.15. The function then generates m exercises, each depending on a few specific abilities. It ensures that each exercise relies on at least one sub-ability, with an average of around 3. Lastly, the function calculates the probability of a correct response for all generated individuals across all generated tasks, using the previously defined formulas. It returns a dataframe where each row represents one individual solving one exercise. The row includes the attributes of both the individual and the task, along with the result, which can be either a probability of correct response or a binary value. With this approach, we can generate as much data as needed, and the calculations ensure that the generated data closely resembles real-life scenarios.

3.5 Implementation

The simulator is implemented in the Python programming language. It exports three functions that correspond to the functionalities outlined in sections 3.1 to 3.3, alongside an abstract dataclass for describing higher-level parameters with multiple implementations, as detailed in sections 3.3.1 to 3.3.3. In the following section, we will examine the configuration and usage of the simulator.

The simple simulator's function is named *simulate_exercise*, and its parameters are the following (using python notation for type and default value):

- *n: int*
- *exercise: list[float]*
- *exercise_weights: list[float] = []*
- *binary: bool = False*

where these correspond to the previously defined inputs accordingly.

The function returns with a tuple of two numpy arrays:

- *abilities_matrix: numpy.ndarray[float]*
- *results: numpy.ndarray[float]*

The parameterized exercise simulator's function is named *simulate_exercise_params*, and extends the previously defined functions input parameters with the following values:

- *params: list[float] = []*
- *params_type: list[Param] = []*

where the first corresponds to the current values of the parameters, while the latter defines the types of parameters used. The type of the output is the same as in the simple case.

The simulator defines an abstract class *Param* to create and use different types of higher-level parameters. The common attributes of the *Param* implementations are the following:

- *ability_index: int*
- *max_change: float*
- *min: float = 0.0*
- *max: float = 10.0*

There are three implementations for the abstract *Param* class: *LinParam*, *PowerParam*, and *ExpParam*, as detailed in sections 3.3.1-3.3.3. The *PowerParam* extends the attribute list with one extra parameter, the exponent, previously denoted as l, which defaults to 2. Therefore, creating a *PowerParam* can be configured like: *p = PowerParam(ability_index=2, max_change=1.5, exponent=3)*.

Lastly the simulator has a *data_synthesis* function. This function's attributes are three numbers, and an optional Boolean value:

- *number_of_people: int*
- *number_of_exercises: int*
- *number_of_abilities: int*
- *binary: bool = False*

This function returns a *pandas DataFrame* where each row represents one individual solving one exercise. The dataframe's columns are: *person_id, exercise_id, ability, exercise, weight, result*, where *ability, exercise,* and *weight* are all k dimensional vectors, k being the *number_of_abilities* the model accounts for.

Conclusions

The developed simulator, along with its underlying mathematical model, is capable of simulating both current cognitive tests and more complex ability-enhancing tasks. It can also generate data suitable for neural network-based models, aimed at uncovering subskill dependencies. Leveraging the simulator, these novel cognitive models can be validated.

Acknowledgements

This research was supported by the Ministry of Culture and Innovation and the National Research, Development and Innovation Office within the Cooperative Technologies National Laboratory of Hungary (Grant No. 2022- 2.1.1-NL-2022-00012).

References

[1] Ha, T. D.: Applying multidimensional three-parameter logistic model (M3PL) in validating a multiple-choice test, International Journal of Scientific and Research Publications, 7(2) (2017) pp. 175-183

[2] Jiang, Y., Yu, X., Cai, Y., & Tu, D.: A multidimensional IRT model for ability-item-based guessing: the development of a two-parameter logistic extension model. Communications in Statistics-Simulation and Computation (2022) 1-13

[3] Thompson, Nathan A. and Weiss, David A.: A Framework for the Development of Computerized Adaptive Tests. Practical Assessment, Research, and Evaluation: Vol. 16, Article 1 (2019)

[4] Weiss, D. J., & Guyer, R.: Manual for CATSim: Comprehensive simulation of computerized adaptive testing. St. Paul MN: Assessment Systems Corporation. (2010)

[5] Han, Kyung T. SimulCAT: Windows software for simulating computerized adaptive test administration. Applied Psychological Measurement 36.1 (2012) pp. 64-66

[6] Benson, Nicholas, David M. Hulac, and John H. Kranzler. Independent examination of the Wechsler Adult Intelligence Scale—Fourth Edition (WAIS-IV): what does the WAIS-IV measure?. Psychological assessment 22.1 (2010) 121

[7] Spearman, C. (1904) General intelligence, objectively determined and measured. The American Journal of Psychology, 15, pp. 201-292

[8] Kovacs, Kristof, and Andrew RA Conway. Process overlap theory: A unified account of the general factor of intelligence. Psychological Inquiry 27.3 (2016) pp. 151-177

[9] Sauce, Bruno, and Louis D. Matzel. The paradox of intelligence: Heritability and malleability coexist in hidden gene-environment interplay. Psychological bulletin 144.1 (2018) 26

[10] Jung, Rex E., and Richard J. Haier. The Parieto-Frontal Integration Theory (P-FIT) of intelligence: converging neuroimaging evidence. Behavioral and brain sciences 30.2 (2007) pp. 135-154

[11] Schneider, W. Joel, and Kevin S. McGrew. The Cattell-Horn-Carroll theory of cognitive abilities. Contemporary intellectual assessment: Theories, tests, and issues (2018) pp. 73-163

[12] Fleishman, Joseph J., and E. Ralph Dusek. Reliability and learning factors associated with cognitive tests. Psychological reports 29.2 (1971) pp. 523-530