A HIGHLY CONFIGURABLE DATA ANALYTICS PIPELINE FOR LEARNING

J.A. Santana, P. Thajchayapong, S. Rugaber, A. Goel

Georgia Institute of Technology (UNITED STATES)

Abstract

Despite the widespread use of configurable data pipelines in enterprise settings, research on architectural considerations of learning analytics remains limited. We propose that an event-driven data analytics pipeline constitutes a Highly Configurable System (HCS) which provides an effective and adaptable framework for educational research. By leveraging a modular architecture with a front-loaded declarative configuration layer, our approach is designed to enhance extensibility, usability and efficiency, allowing domain researchers to conduct and replicate analyses on structured and semi-structured data without extensive code modifications. We demonstrate our architecture by executing the pipeline with data from various learning sessions conducted through the Georgia Tech Online Master of Science in Computer Science (OMSCS) program. Through this effort, we derive configuration templates with varying degrees of specificity, allowing analysis replication for comparable use cases. We also employ a set of metrics for evaluation based on feature modeling analysis to measure the configuration schema's structural complexity and illustrate the likelihood of specific feature arrangements. Our work aims to establish a reusable learning analytics pipeline as part of a data architecture for AI-Augmented Learning (A4L) that can be reconfigured to the evolving demands of learning environments with minimal programming effort.

Keywords: AI-Augmented Learning, Data Architecture, Learning Analytics, Online Education

1 INTRODUCTION

Learning analytics has become a cornerstone of modern educational systems, offering data-driven insights that inform both teaching and learning. By analyzing student behaviors, engagement, and performance metrics, it supports personalized learning pathways, targeted interventions, and improved instructional design [1]. However, despite its promise, the field faces several persistent challenges, especially around fragmented data. Educational data often resides in silos, spread across Learning Management Systems (LMS) and third-party educational tools limiting integration and interoperability. This fragmentation, coupled with data quality issues and privacy concerns, makes deriving meaningful insights difficult [2].

To address these issues, several educational data architectures have emerged, including the Total Learning Architecture (TLA), DataShop, Unizin Data Platform, and My Learning Analytics (MyLA) [3], [4], [5], [6]. While these systems standardize and store data, they often lack a highly configurable analytics engine that supports flexible and replicable workflows for research and instructional use. Recent studies by Donadio [7] and Sghir et al. [8] emphasize the importance of modular, interoperable infrastructures but also underscore their limitations in adapting to real-time, domain specific needs.

In response, Georgia Tech's Design Intelligence Laboratory is developing the *Architecture for AI-Augmented Learning* (A4L) [9] with a focus on meso-learning patterns of learning across intermediate timeframes, such as a semester. Meso-learning relies on Integrated Data Analysis (IDA) [10] to aggregate heterogeneous data sources and reveal behavioral trends, supporting timely pedagogical interventions [11].

To enable this, we propose treating analytics pipelines as Highly Configurable Systems (HCS), which allow dynamic customization of analysis behavior through structured configuration inputs [12]. Unlike traditional Software Product Lines (SPLs) that require compile-time selection, HCSs support runtime variability and adaptability. Techniques from SPLs, such as feature modeling and configuration space analysis, can be adapted to model and constrain analytics workflows within HCS frameworks [13].

Building on this idea, we present a modular, event-driven analytics pipeline designed for educational data. It features a configuration-driven engine for specifying data sources, preprocessing actions, and statistical procedures. Deployed in a serverless cloud environment with containerization for reproducibility, the pipeline integrates seamlessly with visualization platforms. Using data from the Georgia Tech OMSCS program [14], we validate this system through feature variability modeling and use-case replication, demonstrating its flexibility and scalability.

The goal of this paper is to introduce a configurable architecture that automates the preprocessing, transformation, and analysis of educational data as part of A4L. This ensures that research workflows are reproducible across different instances of a domain, for example, across multiple semesters of a course offering AI teaching assistants.

2 METHODOLOGY

2.1 Data

In developing solutions for meso-learning, the scope and complexity of the analytics problem are defined by the number, size, and the number of features they contain in the datasets involved. These attributes influence the choice of tools and techniques. To benchmark our pipeline's capabilities, we reference Donadio (2024) and Sghir et al. (2024), who worked with modular infrastructures and high-dimensional cross-intuitional data, respectively [7], [8]. By comparison, our pipeline was designed to support a wider range of dataset sizes and feature complexities, requiring scalable preprocessing and transformation strategies. *Table 1* outlines the datasets used across domains in this study.

The KBAI-JW domain includes data from the OMSCS Knowledge-Based AI (KBAI) class, which used Jill Watson (JW) as a virtual teaching assistant [15] but has not yet developed into a full-scale conversational courseware [16]. Each instance of the KBAI-JW domain contains three core datasets: *Demographics* (e.g., race, gender and birth), *LMS* (e.g., final grade, student unique identifiers), and Chat *Log* (timestamped conversations between students and JW).

The CogSci-VERA domain draws from the Cognitive Science class that employed VERA for conceptual model building [17], [18]. Its datasets include Demographics, User IDs, VERA Logs (student modeling actions), Threads (forum posts), and Surveys (Likert-scale psychometric data). We note that the Cognitive Science class also uses Jill Watson as a virtual teaching assistant. However, the data used in these experiments pertains only to the use of VERA in the Cognitive Science class.

As shown in Table 1, all dataset collections show a gap between median and maximum sample sizes due to one-to-many relationships from users to logs and threads, highlighting the potential for aggregation. Additionally, while the maximum and median number of features shows a similarly large gap, the high feature count in the KBAI-JW Fall 2023 instance stands out. This discrepancy stems from the composite LMS dataset in this instance, which includes additional platform metadata and survey responses. During execution, a curated subset of features relevant to the target analyses must be selected from each dataset.

While the chat logs in the KBAI-JW domain are virtually semi-structured datasets, they present a uniform structure across samples with finite levels of nesting in its log structure. This allows these datasets to be automatically loaded as tabular structures in the data warehouse, meaning they can be treated as structured data. The Threads dataset in the VERA collection has a recursively nested structure, where the top-level schema can appear inside each sample and even deeper within its nested objects, making it a more complex semi-structured dataset that requires specialized processing. Each collection's composition largely supports structured, hypothesis-driven statistical analysis, though considerable pre-processing is required for instances including datasets like VERA's Threads dataset to enable inference.

2.2 Timeline

The pipeline was developed in three major iterations. Version 1 established the core execution flow, implementing transformations and statistical procedures (e.g., T-Test, Negative Binomial Regression) for the KBAI-JW Fall 2023 dataset. However, each new analysis required cloning the script file of another analysis and modifying it to fit the current use case, making the process inflexible and unscalable. Version 2 introduced a pipeline architecture deployed in AWS. The original script was modularized into an analysis component, refactored to accept dynamic input via configuration payloads, and validated by re-running the KBAI-JW Fall 2023 analysis through the pipeline.

Version 3 focused on validating adaptability and production readiness through four tasks: (1) replicating the KBAI-JW Fall 2023 analysis on KBAI-JW Spring 2024 to test cross-instance consistency, (2) integrating new descriptive statistics analysis to evaluate extensibility within the same domain, and (3) adapting the pipeline to a new domain (CogSci-VERA) tested its cross-domain generalizability by verifying that the existing architecture could automatically reproduce results comparable to those generated by the original, domain-specific implementation.

The design of this infrastructure was guided by multiple factors, including the nature and scale of the data, the overall project scope, the selection of appropriate cloud services, and the criteria used to assess the system's flexibility.

Table 1. Composition of Dataset Collections.

Domains	*Dataset Collections (Instances)*	*Datasets Count*	*Dataset Scale (Max | Median # of Samples)*	*Dataset Complexity (Max | Median # of Features)*	*Data Types*
KBAI-JW	KBAI-JW Fall 2023	3	3706 | 598	204 | 9	Structured
	KBAI-JW Spring 2024	3	17970 | 611	40 | 7	Structured
CogSci-VERA	CogSci-VERA Summer 2023	5	68682 | 160	58 | 11	Structured, Semi-Structured

2.3 Architecture

The architecture is defined as a *state machine* that initiates when an event occurs in the cloud environment, as depicted in *Figure 1*. In our AWS implementation, the event is defined as a time-based trigger, often referred to as a scheduled job, that activates the state machine at regular intervals (e.g. daily at 9AM).

Upon being triggered, the state machine receives its input in the form of a payload. This input object defines the instructions and parameters needed for that run. Once received, the state machine proceeds to execute a sequence of code modules accordingly.

The **data fetch module** makes the relevant datasets, specified in the payload, accessible to the downstream components in the pipeline. It achieves this by executing a query to the data warehouse with an *UNLOAD* command that compresses the data in an efficient, machine-readable format (Parquet) and stores it in a data store that holds these files temporarily. The data fetch module receives the status of the UNLOAD command as a response. It will then repeatedly ask for status updates in fixed intervals through polling until it gets a "Complete" status, along with file locations for each of its datasets. The pipeline proceeds to forward this information to the next module.

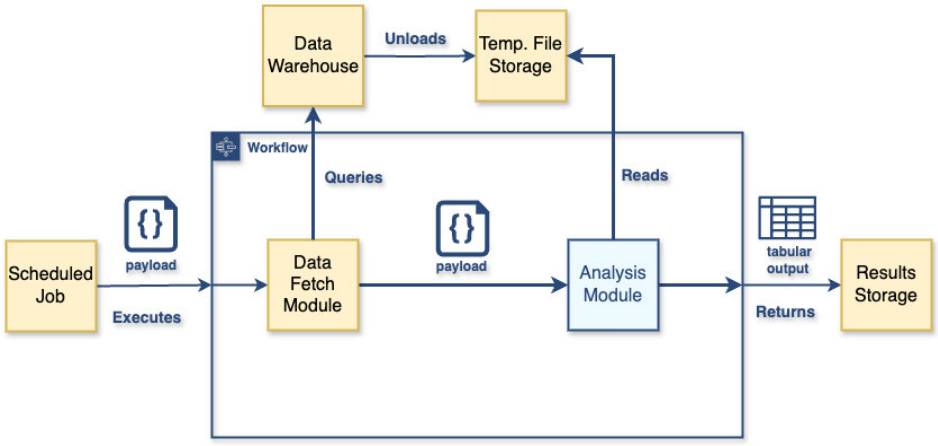

Figure 1. A high-level diagram of the data analytics pipeline

The **analysis module** is the core logical component of the pipeline. Its purpose is to produce analysis results based on the input configuration. As *Figure 2* lays out, this module loads the datasets from the data store, cleans the data through specific pre-processing functions listed in the payload, applies transformations to the datasets as indicated on the payload, and executes statistical procedures for the desired experiment. The analysis results are compiled in a structured format and uploaded to a permanent data store in the cloud environment for other systems, such as for data visualization, to make use of them.

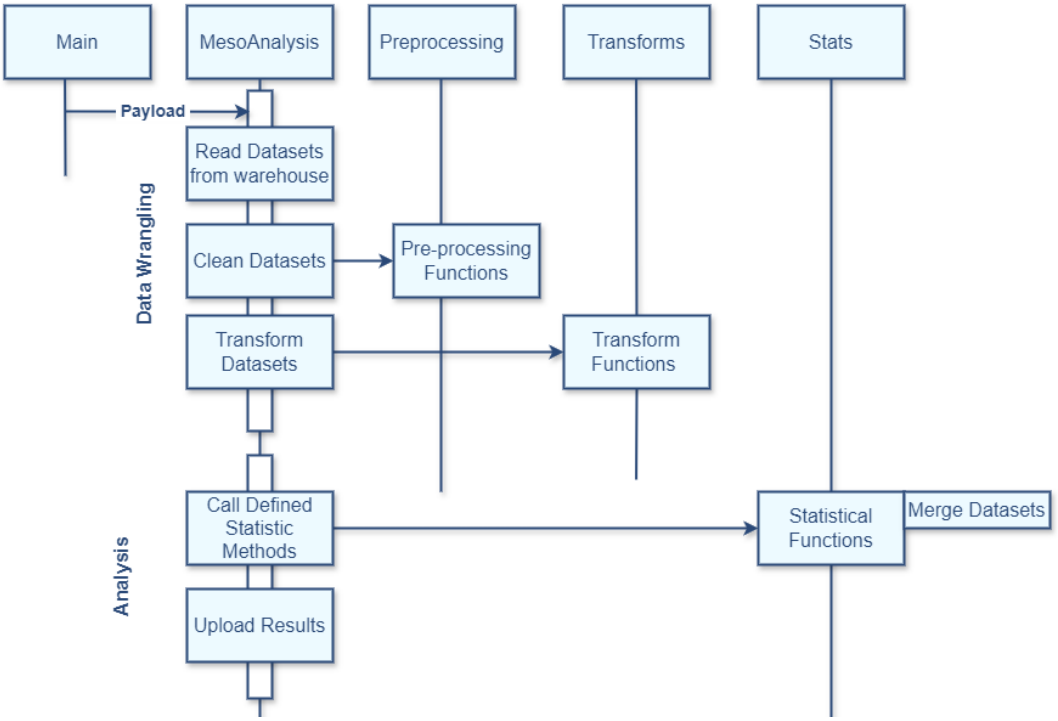

Figure 2. A sequence diagram of the analysis module

The payload is a critical component of the pipeline that provides a declarative interface through which researchers specify analysis workflows. Structured as a JSON configuration file, this configuration shapes the effective behavior of the pipeline.

In this project, the payload for the KBAI-JW Fall 2023 instance served as a baseline from which other payloads were derived. Toward the end of the project, a general-purpose payload template was created to formally capture the available attributes in a more reusable structure. The template can be populated to prepare an input configuration for any new analysis. Alternatively, for new instances within the same domain (e.g. KBAI-JW), it is more efficient to duplicate and modify the existing payload, requiring only moderate changes. The data model diagram in *Figure 3* captures the structure and valid attributes, describing the full shape and semantics of this configuration.

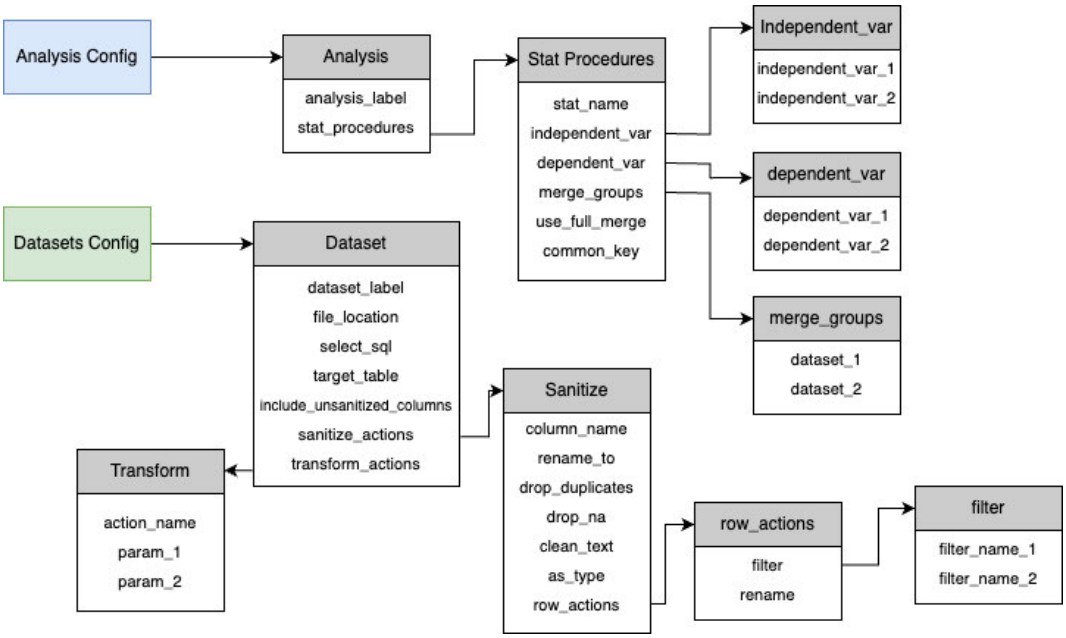

Figure 3. The payload's data model.

3 ANALYSIS METHODOLOGY

3.1 Representation

While the pipeline's payload is formatted as JSON for execution, our analysis focuses on its flexibility from the perspective of HCS, requiring a specialized representation. To this end, we adopt feature models, a hierarchical format that expresses the configuration's key-value pairs as *features.* A feature refers to a distinctive characteristic of a system [19], which aptly describes the parameters and procedures used in our payload.

In standard feature models, each feature is treated as a Boolean property, indicating whether it is selected in a given configuration. Features are grouped under parent features according to specific relationships, labeled as mandatory, optional, alternative (logical XOR, one feature must be selected) or some-out-of-many (logical OR, one or more features may be selected).

Feature models can also express *constraints*, which are logical conditions that affect feature combinations. While feature modeling is most commonly associated with SPLs, it also provides a helpful framework for analyzing flexibility in HCS.

To encode feature models, we use the Universal Variability Language (UVL), a community-driven standard for feature model representation in a consistent, unified format [20]. In our analysis, UVL captures the structure of the payload and its supported options, serving as a foundation for formal variability analysis.

Figure 4 and 5 show how our earlier data model translates to a feature model representation. In this visualization, filled circles represent mandatory features, empty circles represent optional ones, filled half-circles indicate OR groups and empty half-circles indicate XOR groups. While this model is a simplification of what the payload can encode in raw JSON, it accurately captures the complete space of supported and validated configurations by the data analytics pipeline.

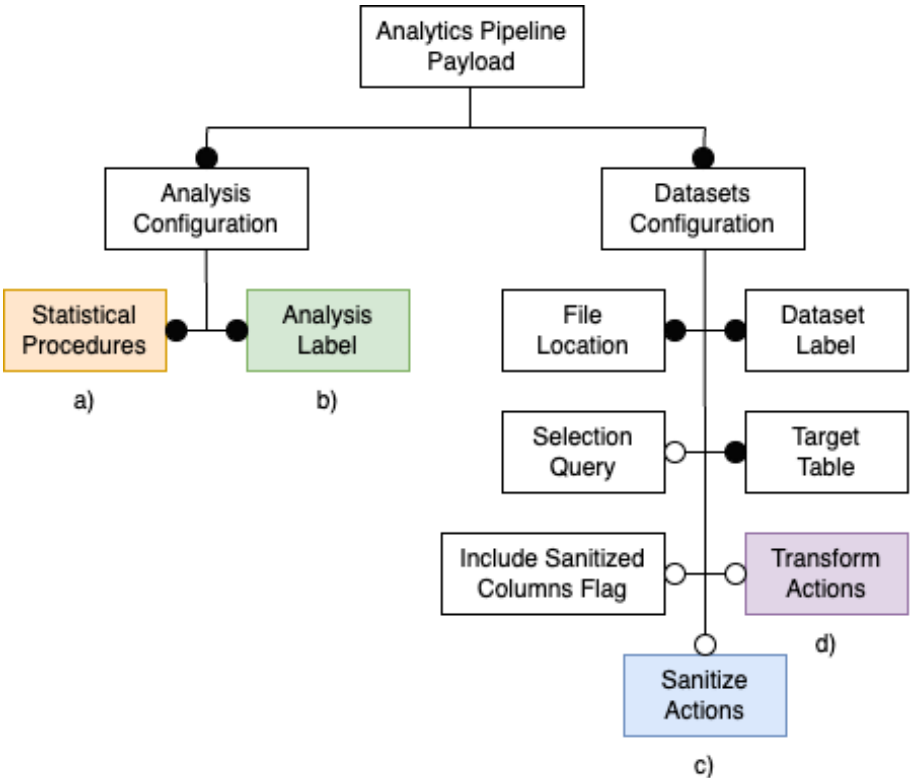

Figure 4. The feature model mapping the payload.

There are several terms that are important to understand feature modeling in this context. A selection of enabled features is called a *configuration*. A configuration is considered *valid* if all structural rules and constraints are satisfied. A *variant feature* is any feature that is not always present, which includes optional features, any features in OR or XOR groups, or features excluded through constraints. For

example, a rule like “if feature A is selected, then feature B must also be selected” affects the likelihood of feature A being selected. Variant features are very notable as they are directly connected to variability metrics, which we use to gauge our payload’s flexibility.

Constraints can involve features at different levels of the hierarchy, aside from parent-child or sibling relationships. These are called *cross-tree constraints*. When constraints become overly restrictive, they can produce undesirable states for our features. Some outcomes include introducing *dead features*, which can never be selected in any valid configuration, or *false-optional features,* which are marked as optional in the model but are always included due to constraints, effectively behaving like mandatory features.

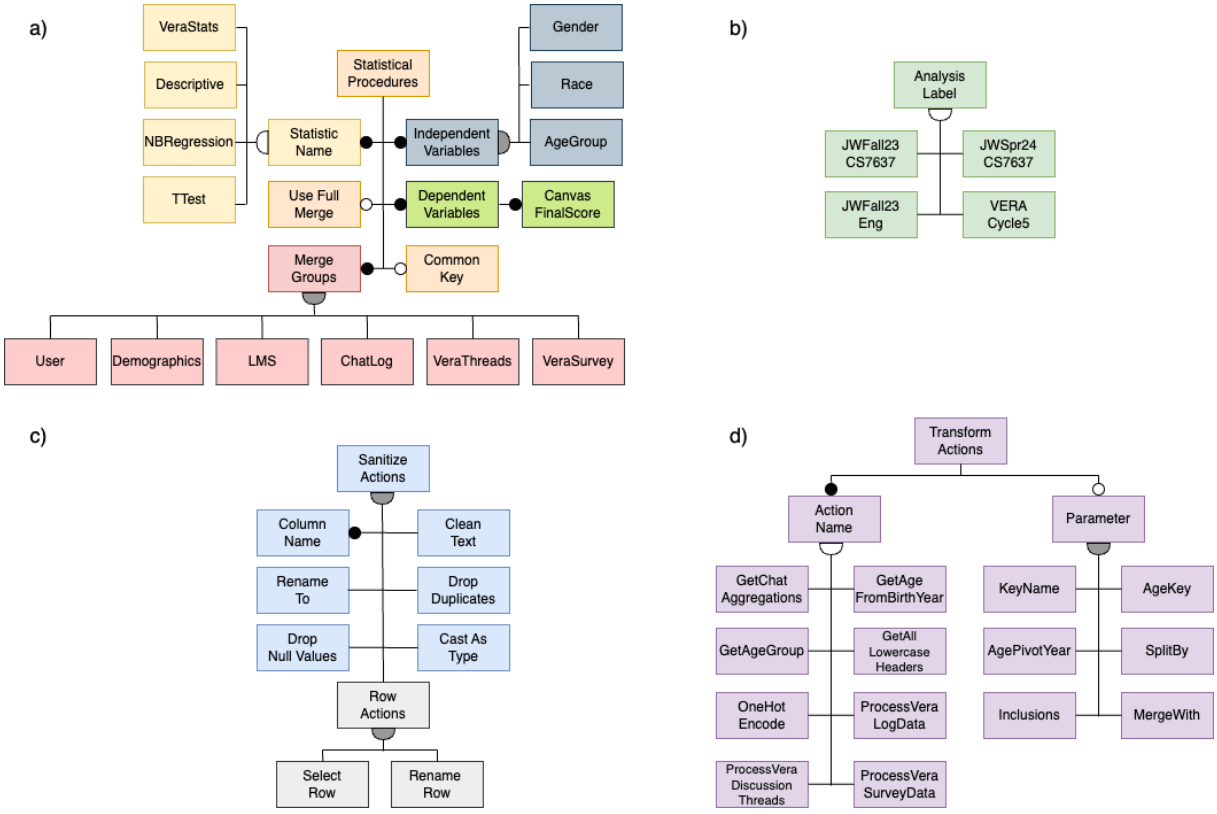

Figure 5. Extension sections of the feature model mapping the payload shown in Figure 4.

Since constraints can significantly reduce the space of valid configurations in elaborate ways, analyzing feature models becomes non-trivial. This complexity is why specialized tools are necessary for extracting insights from the configuration space.

3.2 Tooling

Our analysis is executed in Python 3.12 using FlamaPy for Feature Modeling Analysis. FlamaPy is an open-source Python framework designed to facilitate the modeling and automated analysis of SPLs and HCSs [21]. It leverages SAT and BDD under the hood, two frameworks used to analyze complex Boolean decision problems [22]. While similar tools use XML or other more obscure languages for feature modeling, FlamaPy primarily uses UVL as its representation of choice. FlamaPy enables access to a variety of functions to explore commonality, variability and combinatorics in a chosen feature model. These functions are discussed in the next section and used to produce the analysis results. FlamaPy is particularly valuable when working with configurable systems, as it enables validation of configuration models, quantification of flexibility, and systematic evaluation of design trade-offs.

3.3 Feature Modeling Metrics

Feature Modeling enables the analysis of both the structural composition and probabilistic behavior of a configuration space. In addition to basic statistics such as the number of features and the number of valid configurations, it allows us to quantify configurability in formal terms through *variability*. Variability is defined as the ratio of valid configurations to the number of possible configurations. This definition yields two related but distinct metrics; *Total Variability (V)* measures the fraction of valid configurations relative to the entire theoretical configuration space, assuming no constraints. On the other hand, *Partial Variability (V_p) restricts* possible configurations to only *variant features*, providing a more precise estimate of flexibility by isolating the effect of constraints and disregarding fixed features [23]. Because it focuses on the features that contribute to actual configurability, partial variability is generally considered a stronger indicator of a system's flexibility.

Another key metric is *homogeneity (H)*, which captures how uniformly features appear across all valid configurations [23]. A value of $H \approx 1$ suggests that most features are fixed (mandatory or dead), indicating a rigid, low-variability model. Conversely, a value at $H \approx 0.5$ means every feature is selected in about half of the configurations, implying all features are variant and the model exhibits maximum variability. Collectively, these metrics provide a multifaceted understanding of a feature model's structure and variation potential, offering insights into how constraints shape the configuration space.

3.4 Benchmarks

To validate our claim that the pipeline qualifies as an HCS, and thus, a flexible architecture suited to the demands of educational data analytics, we conducted a benchmark comparison across three broad metric categories, evaluated over four models. The results are presented in *Table 2*. The Structure & Scale category includes counts of total and variant features and total versus valid configurations. The *Variability* section includes both Total and Partial Variability, along with homogeneity. The *Correctness* category evaluates the model integrity, capturing the number of dead features, false optional features, and structural conflicts, all of which ideally approach zero.

We computed these metrics using FlamaPy, focusing on the pipeline's payload as the central determinant of flexibility in our system, compared against three other reference models. From the FlamaPy web tool, we selected the Xiaomi Smartband 8 model, which captures the configuration space of a smartwatch product line [24]. The remaining two models, Berkeley DB and axTLS, were sourced from the UVL repository. Berkeley DB represents the configuration space for Oracle's open-source embedded database library, while axTLS refers to the lightweight transport layer security (TLS) tailored for embedded and resource-environments [25], [26]. Both are recognized in the literature as highly configurable systems, making them suitable models to be used for comparison in this benchmark.

4 RESULTS

Our analysis results in Table 2 show that the A4L Data Analytics Pipeline (DAP) strikes a healthy balance between complexity and usability. With 66 configurable features (49 which are variant features), it offers a significant number of options without excessive intricacy. While our system supports fewer possible configurations than the more mature technical systems, it outranks the other models in *partial variability,* meaning the variant (optional) features that researchers might want to tune are more adaptable compared to its counterparts.

The homogeneity score of 0.608 indicates our system achieves a healthy equilibrium between flexibility and consistency. On a scale where 0.5 is complete unrestricted flexibility and 1 is complete rigidity, this score indicates that while features are very permissible, there is enough structure to keep analyses coherent.

The payload's feature model performs well in most correctness checks. It contains no dead features, meaning every feature participates in at least one valid configuration. There is only one false-optional feature, *CommonKey*, which is flagged as optional, yet it's consistently included in all actual configurations for the purposes of IDA. Compared to other models such as axTLS, with 11 dead feature and two false-optional features, the DAP payload exhibits a healthy configuration profile.

Table 2. Feature Modelling Results.

Category	Metric	A4L DAP Payload	Xiaomi SmartBand 8	Berkeley DB	axTLS
Structure & Scale	Total Features	66	37	76	96
	Variant Features	49	25	75	61
	Estimated Total Configurations	$7.413e9 \times 10^{9}$	32256	1.150×10^{14}	5.655×10^{14}
	Valid Configurations	1.237×10^{9}	21	4.080×10^{9}	8.262×10^{11}
Variability	Total Variability	1.677×10^{-11}	1.528×10^{-10}	5.400×10^{-14}	1.043×10^{-17}
	Partial Variability	2.198×10^{-6}	6.258×10^{-7}	1.080×10^{-13}	3.583×10^{-7}
	Homogeneity	0.608	0.628	0.724	0.561
Correctness	Dead Features	0	0	0	11
	False Optional Features	1	0	2	2
	Conflicts/Errors	0	0	0	0

Figure 6 displays the distribution formed by the number of variant features plotted against the count of valid configurations for that many variants. The curve is approximately normal, peaking at 37 features per configuration. In absolute terms, the most common region, roughly 6 to 7 million configurations, represents about 1% of the entire valid configuration space. This indicates that researchers and other practitioners would be likely to gravitate to a relatively centralized band of features. This behavior matches the model's homogeneity score, meaning roughly 60% of variant features appear in almost every configuration, while the remaining 40% appear occasionally in specialized use cases. Together, the distribution and the previous metrics reveal a configuration landscape that promotes a shared baseline while leaving enough room to accommodate domain-specific extensions.

The model remains scalable while avoiding the brittleness that accompanies larger feature models. Its moderate complexity and sound structure provide an optimal balance of generality and specificity for educational data analytics workflows, making it a strong foundation for future studies and evolving educational technology applications.

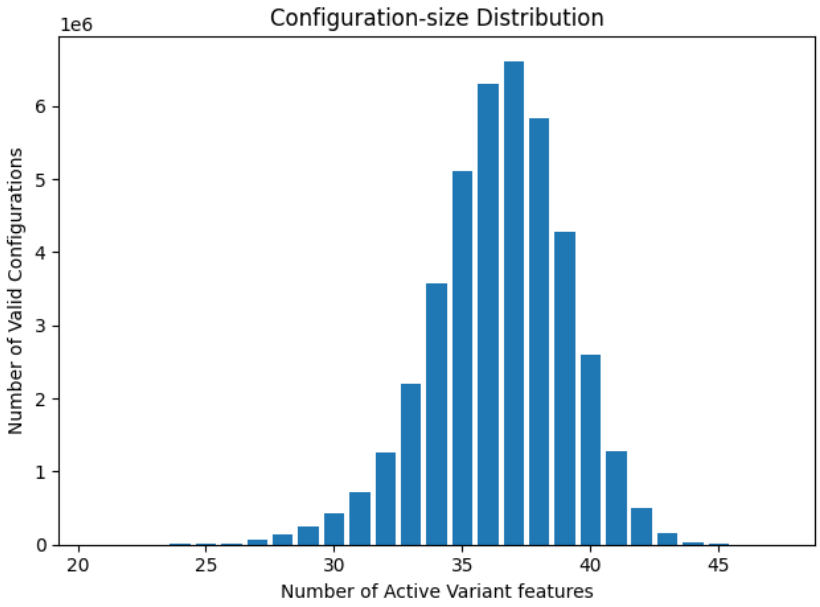

Figure 6. The payload's configuration-size distribution.

5 CONCLUSIONS

By analyzing the pipeline as a Highly Configurable System (HCS), we demonstrated how a modular, event-driven architecture with front-loaded declarative configuration can enhance extensibility, flexibility and efficiency in educational data analysis workflows.

Our feature modeling analysis revealed that the data analytics pipeline achieves a strong balance between flexibility and structural coherence. With 66 total features (49 variant) and approximately 1.24×10^{9} valid configurations, the pipeline maintains significant variability while avoiding the excessive complexity found in other mature, highly configurable systems. The comparatively high partial variability score (2.20×10^{-6}) further validates our approach, indicating that the system successfully focuses variability on the optional features that enable customization for specific analytical needs, where they are most valuable.

While our implementation demonstrates the value of a configurable analytics pipeline as part of a data Architecture for AI-Augmented Learning (A4L), several opportunities for improvement remain. In particular, the architecture could benefit from further decoupling of transformation and statistical analysis components, as exemplified by the VERA adaptation in Task 3. Exploring distributed computing patterns may support this goal but would require careful consideration of how added complexity affects the payload configuration and the constraint space of the feature model. Ensuring the system retains a balance of flexibility and stability will be necessary.

A key limitation of our current modeling approach is the lack of cardinality-based feature modeling analysis. Although traditional feature models sufficed for our project scope, cardinality representations will be increasingly important as the pipeline scales. Continued development in this area will be crucial for more robust HCS evaluation. Further work should also include direct comparison to other data pipelines or ETL systems to better contextualize strengths and limitations of our architecture.

ACKNOWLEDGEMENTS

This research was supported by NSF Grants #2112532 and #2247790 to the National AI Institute for Adult Learning and Online Education. We thank members of the A4L team in Georgia Tech's Design Intelligence Laboratory for their contributions to this work.

REFERENCES

[1] G. Siemens and P. Long, "Penetrating the fog: Analytics in learning and education," EDUCAUSE Review, vol. 46, no. 5, pp. 30–40, 2011.

[2] R. Ferguson, "Learning analytics: drivers, developments and challenges," International Journal of Technology Enhanced Learning, vol. 4, no. 5–6, pp. 304–317, 2012.

[3] "Modernizing Learning: Building the Future Learning environment," ADL Initiative. Accessed: May 12, 2025. Retrieved from http://0.0.0.0:4000/publications/2019/04/modernizing-learning/

[4] K. R. Koedinger, R. S. J. d Baker, K. Cunningham, A. Skogsholm, B. Leber, and and J. Stamper, "A Data Repository for the EDM Community: The PSLC DataShop," in Handbook of Educational Data Mining, CRC Press, 2010.

[5] Unizin, Knowledge Base, Accessed: May 12, 2025. Retrieved from https://unizin.org/knowledge-base/

[6] "My Learning Analytics," U-M Information and Technology Services. Accessed: May 12, 2025. [Online]. Retrieved from https://its.umich.edu/academics-research/teaching-learning/my-learning-analytics

[7] M. Donadio, "Toward Modular Analytics Infrastructure for Personalized Learning," Journal of Learning Analytics, vol. 11, no. 1, 2024.

[8] M. Sghir, I. Azough, and A. Amine, "Interoperability Frameworks for Learning Analytics Platforms: A Systematic Review," Educational Technology Research and Development, vol. 72, no. 1, pp. 123–145, 2024.

[9] A. Goel, P. Thajchayapong, V. Nandan, H. Sikka, and S. Rugaber, "A4L: An Architecture for AI-Augmented Learning," in Online Higher Education Administration: Strategies for Maximizing Returns and Improving Learning Outcomes, K. S. Ives, R. Schroeder, and M. Cini, Eds. New York, NY: Routledge Press, in press, 2025

[10] P. J. Curran and A. M. Hussong, “Integrative data analysis: The simultaneous analysis of multiple data sets,” Psychological Methods, vol. 14, no. 2, pp. 81–100, 2009, doi: 10.1037/a0015914.

[11] D. Gašević, S. Dawson, T. Rogers, and D. Gašević, "Learning analytics should not promote one size fits all: The effects of instructional conditions in predicting academic success," The Internet and Higher Education, vol. 28, pp. 68–84, 2015.

[12] S. Apel, D. Batory, C. Kästner, and G. Saake, Feature-oriented software product lines: Concepts and implementation. Berlin: Springer Science & Business Media, 2013.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study. Pittsburgh, PA: Carnegie Mellon University Software Engineering Institute, 1990.

[14] Joyner, D. A., Goel, A., Isbell, C., & Starner, T. (2019). Five Years of Graduate CS Education Online and at Scale. In Proceedings of the 2019 ACM Global Computing Education Conference (CompEd) (pp. 1–8). ACM.

[15] Goel, A. and L. Polepeddi (2018) Jill Watson: A Virtual Teaching Assistant for Online Education. In Learning Engineering for online education: Theoretical contexts and design-based examples, C. Dede, J. Richards, & B. Saxberg (Editors). New York: Routledge

[16] Kakar, S., Basappa, R., Camacho, I., Griswold, C., Houk, A., Leung, C., Tekman, M., Westervelt, P., Wang, Q., & Goel, A. (2024) SAMI: An AI Actor for Fostering Social Interactions in Online Classrooms. In Procs. of 20th International Conference on Intelligent Tutoring Systems (ITS 2024).

[17] An, S., Rugaber, S., Weigel, E., & Goel, A. (2021) Cognitive Strategies for Parameter Estimation in Model Exploration. In Procs. 43rd Annual Conference of the Cognitive Science Society.

[18] Fryer, K., Thajchayapong, P., & Goel, A. (2025). AI Adoption in Education: How Cognitive, Motivational, and Demographic Factors Shape Help-seeking with a Virtual Teaching Assistant. To appear in Procs. 17th International Conference on Education and New Learning Technologies (EduLearn 2025).

[19] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study:,” Defense Technical Information Center, Fort Belvoir, VA, Nov. 1990. doi: 10.21236/ADA235785.

[20] D. Benavides, C. Sundermann, K. Feichtinger, J. A. Galindo, R. Rabiser, and T. Thüm, “UVL: Feature modelling with the Universal Variability Language,” Journal of Systems and Software, vol. 225, p. 112326, Jul. 2025, doi: 10.1016/j.jss.2024.112326.

[21] J. A. Galindo, J.-M. Horcas, A. Felferning, D. Fernandez-Amoros, and D. Benavides, “FLAMA: A collaborative effort to build a new framework for the automated analysis of feature models,” in Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B, in SPLC ’23, vol. B. New York, NY, USA: Association for Computing Machinery, Aug. 2023, pp. 16–19. doi: 10.1145/3579028.3609008.

[22] G. Audemard and L. Sais, “SAT based BDD solver for quantified Boolean formulas,” in 16th IEEE International Conference on Tools with Artificial Intelligence, Nov. 2004, pp. 82–89. doi: 10.1109/ICTAI.2004.106.

[23] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, “FLAME: FAMA Formal Framework (v 1.0)”.

[24] “FlamapyIDE.” Accessed: May 13, 2025. [Online]. Retrieved from https://ide.flamapy.org/

[25] “Oracle Berkeley DB: Storage.” Accessed: May 09, 2025. [Online]. Retrieved from https://www.oracle.com/technical-resources/articles/database/oracle-berkeley-db-storage-layer.html

[26] “axTLS Embedded SSL.” Accessed: May 09, 2025. [Online]. Retrieved from https://axtls.sourceforge.net/