Deep System Instrumentation for In Situ Human-AI Interaction Measurement within Complex Information Systems

Joshua C. Poore[1], Alex Veerasammy[1], Amir Ghaemi[1], Grant Tamrakar[1], Kelsey Rassmann[1], & Craig Lawrence[1,2]

[1] Applied Research Laboratory for Intelligence and Security (ARLIS), University of Maryland, College Park, Maryland
[2] Institute of Systems Research, University of Maryland, College Park, Maryland

Abstract

In this report, we explore key objectives for Human Systems Integration (HSI) Test Evaluation Validation and Verification (TEVV) for human-Artificial Intelligence (AI) interactions within analytical and information systems. Owing to their features and function, these systems pose challenges to the test community in evaluating the reciprocal influence of human and AI elements on one another. We argue that software system instrumentation provides the necessary capabilities to resolve measurement gaps. We further report our own efforts in adapting this technology for HSI TEVV and making these capabilities accessible to the larger test community.

Introduction

AI is both a smaller and a bigger endeavor than it was at its inception. In pragmatic practice, AI is smaller in that 'general AI is neither the singular objective for the field, nor is it the desire for most organization. Rather, industry develops modular, functionally focused AI and ML capabilities as services, deployable into larger platforms (e.g., social media, ecommerce), or enterprises (e.g., integrated business operations). But, in this way, AI is also a bigger endeavor because the influence and complexity that small components add within platforms and enterprises can be dramatic, particularly where human users (and behavior) are important elements (or inputs) into platforms.

The influence of AI and the complexity it adds at scale is laid bare in analytic and information systems, where there is a fervor for AI to deliver processed information and recommendation to analysts from a wide variety of sectors. Indeed, AI within this context presents a unique challenge. Unlike content delivery platforms, where AI's role is to brusquely deliver content and advertisement to users based on preference models, AI in analytical and information systems delivers processed information to user to support human reasoning and decision-making. Beyond questions of accuracy and precision, AI in this context, more than other digital platforms, can shape how users interpret other data and holistically understand and synthesize larger sets of data. In turn, decisions made within these systems based on information processed and exploited by AI can have profound effects on sectors in which it operates—business, finance, and intelligence. The implicit challenge regarding AI in this context is gauging reciprocal influence between users and AI and ascertain whether AI exerts undue influence in synthesis over other information. It is essential to effectively manage this influence so that it is appropriately calibrated against the precision and accuracy of AI.

Largely owing to Lee and See (2004) there is a thriving literature and set of subjective batteries for understanding trust in autonomy and AI. This provides a theoretical foundation for understanding the influence of AI within information systems as described above. However, the desire to deploy AI and ML into larger, distributed systems poses a challenge for traditional measurement approaches to HSI TEVV. These are slow moving endeavors and tend rely on subjective measurement, which do not translate well for informing software development and system management. As such, there is a need to understand how AI influences human users to understand if AI delivers expected value to operations and operational end-users. This requires three things: 1) *in situ methods for capturing human AI interactions at a transactional, behavioral level* that can be translated into actionable development objectives; 2) *traceability* on both ends of the human-system transaction—initiation of user behavior and origin of information (e.g., AI/ML service); 3) *granularity in measurement*—the ability to widely sample behavior as users interact with Analytical applications, ho-

listically. High temporal resolution and deep metadata context on each behavior serves the inferential capabilities that make gauging reciprocal human-AI influence possible.

To meet the needs for in situ human-AI interaction measurement, we are developing measurement apparatus from adapted, commercial off-the-shelf (open source) software instrumentation technology, supportive processing, and analytical capabilities that meet the requirements for HSI TEVV delineated above. This report articulates how we address gaps left by COTS technology and analytical lessons learned from past work. We also articulate our approach for processing instrumentation data and producing developer-actionable insights based on descriptive and inferential use of graph analytics. We discuss at length why graph methods, coupled with advanced segmentation practices provide the basis for an inferential framework for AI influence that is widely applicable to a range of analytical and information system use-cases, and why this approach reconciles past lessons-learned from other approaches. Finally, we discuss how our efforts will provide the larger test community scaffolding and best practices, as well as access to methods and apparatus that have had steep skill requirements, historically. Building our on open-source development, we believe the test community will better be able to expose observable patterns of human (and system) behavior that can be used to validate and refine other important measurement objectives, e.g., trust in autonomation, and allow for aligning such observations against operational outcomes to identify issues addressable with engineering solutions.

Human-AI Interaction Test Objectives within Analytical and Information Systems

We use 'Analytical and Information Systems' to refer to a specific set of distributed systems; their purpose is to create information or intelligence, and sometimes consume information and intelligence for other purposes (e.g., automation). In most embodiments, they incorporate search and reference function (e.g., search, query, and filter for data or information), summarization and some inferential features (e.g., statistical operations), and an information-rich user interface, such as an interactive 'dashboard'. Notable examples in the public consciousness are Palantir products, various financial planning products, Geographic Information Systems (GIS, e.g., ESRI products), and business intelligence applications (e.g., Apache Superset (**Figure 1**)).

The most discriminating feature of Analytical and Information Systems is that they are *augmentation technologies* designed to assist human decision-making and risk analysis. Often, they incorporate autonomous elements (e.g., specialized processing on specific data feeds, AI-based predictive capabilities), however, they are designed to provide human users data and information both, with the assumption that the (human) user provides synthesis and makes decisions with operational impacts, based on that synthesis.

The objective for test and evaluation of human-AI interaction within analytical and information systems is to understand the influence of human and AI elements on one another, their mutual influence on the larger system, and on the workflows for which that system was created to support. These objectives are very similar to a more general view of HSI TEVV. However, analytical and information systems that incorporate AI make these objectives especially challenging because of their nature. Again, these systems provide both information and scaffolding to human users to enable a greater, holistic synthesis of available information. This means that an evaluation of reciprocal influences between human and AI elements in the system context requires not only an examination of how humans utilize and rely on AI components, but also how AI may augment or contaminate human users' interpretation of the larger information environment. Where the laboratory provides some tools to mount such an examination, it is not designed for the agility and alacrity with which AI can be developed and deployed into modern information systems and larger platforms.

Figure 1. Apache Superset Business Analytics Dashboard
© Apache Software Foundation 2022

How We Test Today

How we perform HSI TEVV for Analytic and Intelligence systems today is not significantly different than it was a decade ago. HSI is "the management and technical discipline of planning, enabling, coordinating, and optimizing all human-related considerations during system design, development, test, production, use and disposal of systems, subsystems, equipment and facilities" (SAE 2019). It owes to a long-standing tradition of safety and survivability testing in control and autonomous systems originally formalized by the US Department of Defense (DoD). It borrows methods from a wide range of disciplines: human factors, human computer interaction, psychology, systems engineering, and organizational research, among others.

Stemming from a test tradition for evaluating physical

systems (e.g., aeronautical, naval, tactical, and robotic), the field's capabilities for evaluating distributed software systems did not grow commensurately with the sophistication and accessibility of technology for standardizing and managing deployed software at scale (e.g., containerization). As such, there remains methodological mismatch on both scale and speed: First, current mainstay ethnographic, subjective survey, and laboratory assessments are ill-equipped to sample system behavior (including user behavior) within ecological valid scales, which limits an understanding the scope and severity with which new components influence the system and incumbent information environments. Second, a regimen of systematic, controlled laboratory studies can generate meaningful findings about how new information can influence synthesis and decision-making within a larger information environment. However, particularly for large scale, production systems (not prototypes) laboratory testing cannot meet the agility of modern software system development and deployment practices.

Software System Instrumentation

Software system instrumentation refers to set of methods for reporting on the state of and actions with a software environment, or larger system. Instrumentation practices vary widely and suite a wide range different purposes—access auditing, error messaging, system telemetry, and behavioral usage logging. Most germane to this report are Behavioral Usage Logging and System Telemetry. The former reports on user interactions (e.g., 'click', 'mouseover') with specific elements (e.g., 'div', 'form'); virtually all programming languages that service user interface development provide classes uniquely suited to the practice. Behavioral Usage logging is most recognizable in the context of web analytics (see **Figure 2**) where it has been a ubiquitous practice since the early 2000's and is currently the foundation for a multi-billion-dollar industry lead by Google Analytics.

System Telemetry uses specific frameworks (e.g., Prometheus, Zipkin, Jaeger, Open Telemetry) to provide traces between distributed services and applications (including clients) to report on the path through which requests and response propagate through the system (or platform), as well as metadata (e.g., time to return). Prometheus, for example, is an incumbent of the Cloud Native Foundation.

Both technologies—Behavioral Usage Logging and System Telemetry—are in wide use on large scale web platforms. Importantly, these technologies *could* jointly collect features essential to evaluating the reciprocal influence between human and AI elements of larger systems—but they are not currently used to this purpose.

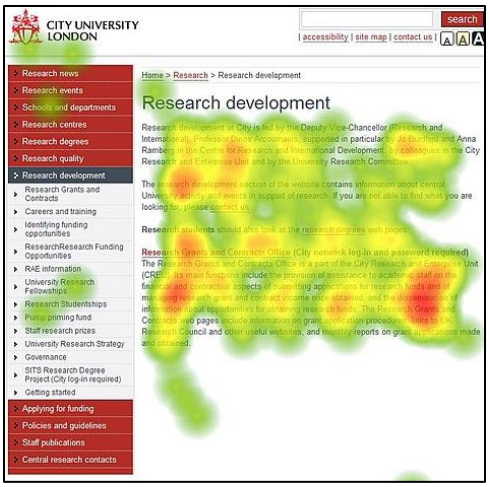

Figure 2. Heatmap of User Activity (Web-Analytics). Image Credit: City University Interaction Lab; License: https://creativecommons.org/licenses/by/2.0/deed.en

Indeed, current (web) instrumentation technology has the requisite capabilities required for capturing high resolution behavioral transactions between human and machine elements of large distributed systems and do so with both traceability and granularity. However, there remain gaps in the practice for applying these technologies for the purpose of understanding human-AI reciprocal influence, even though the technologies are capable.

First, Behavioral Usage Logging and System Telemetry generally serve different purposes. As such, COTS products generally do not provide comprehensive, integrated solutions that allow for purposing both capabilities for HSI testing; there are gaps in the practice of integrating these technologies in efficient ways that reduce redundancies and allow for dual use. Second, and most notably, there are gaps in established best practices for analytical pre-processing that support more sophisticated behavioral analysis (e.g., reciprocal inference). These gaps largely owe to the very simple point that HSI testing doesn't help monetize applications in the same way that web-analytics do and the service industry that has built itself around web/business analytics doesn't have any incentive to provide HSI testing as a service or put tradecraft in the public domain. Because of these gaps, there are examples for analytical approaches for more sophisticated behavioral analyses, but few essential pragmatic resources for collecting the right data to feed these approaches and little guidance on how to effectively treat data so that it can be submitted to these approaches.

Lessons Learned

Our objective is to address the gaps outlined above so that

software system instrumentation technologies might be applied to the deeper challenge of understanding reciprocal influence between human and AI in analytical and information systems. We are not the first to attempt this (for some of us, this is not our first attempt). Historical lessons learned provide illustrative guidance on how best to address technical gaps and gaps in practice both.

Instrumentation Leads Analytics. Early approaches to understand user behavior within the context of Analytical and Information systems took a nomothetical approach to understanding granular user behavior. This is to say that for every action and analyst might perform, that action would be tagged or correlated to some cluster pertinent to an analytical process (c.f., Alonso & Li 2005; Wright et al 2006; Poore 2017). These approaches were primarily driven by triage use-cases—by developing a model of analysts' workflow and correlating actions/events logged through application clients, they attempt to predict and deliver content to analysts based on the workflow "state" they were observed in and the content they interacted with.

These "top down" modeling approaches proved more effective for application design than for behavioral analysis; theoretically driven models prove brittle in attempting to generalize models across applications. Pragmatically, the level of effort associated in explicitly associating model facets or states with specific user behavior across different applications is prohibitive. A key lesson learned in this case is that coercing instrumentation practices to serve specific analytical approaches is costly, prone to adoption risks, and results in brittle models. As such, our approach is to adapt processing and inferential methods to use canonical instrumentation data. Relying on already ubiquitous instrumentation methods provides a dual-use value proposition: developers gain the benefit of standard usage analysis and system telemetry, while the testing community co-opts the same data for different purposes.

Preprocessing has a Greater Cost than Analytics. Instrumentation data generally takes for the form of flexible message formats, and a single user session can create thousands of such messages. Even simple operations like filtering through messages can be complex and time consuming and require a deep skillset in data science and data engineering. Like so many endeavors that require data scientists (not just analysts), processing instrumentation data follows a Pareto principle—80% of the work is in preprocessing and 20% of the work is the analysis itself.

The biggest barrier to entry for utilizing Software System Instrumentation for complex behavioral analytics is data preparation and preprocessing. Instrumentation data, especially Behavioral Logging and System Telemetry, is challenging to work with because it uses very few formatting standards, by design. To reduce the memory costs associated with generating logs and shipping them, Instrumentation packages utilize very simple key-value message structures to capture data. This also allows for customization, which is valuable. However, this creates several challenges in a 'dual-use' model. That every message doesn't conform to the same schema (i.e., different keys), creates challenges for search and reference in analytical pipelines.

Analytic and information systems result in challenging instrumentation data to work with. Where ecommerce and content delivery system workflow are, by design, linear in nature, analytical software designed for information synthesis result in organic user workflows that are highly varied and highly recursive (analyst may repeat the same operations with different search terms, for example). As such, identifying discrete workflows and tasks within instrumentation data can require significant skill in forensic data science and data engineering.

To bootstrap a community of practice around software instrumentation data for deep behavioral analysis, there is a greater need for practices and utilities that make the data easy to manage, curate, query, filter treat than for analytics themselves. Our work aims to bootstrap extensible preprocessing that increase the accessibility of instrumentation data to the wide set of disciplines within the test community.

Analytical and Intelligence Applications have Intrinsic, Behavioral Properties. Analytical approaches for understanding reciprocal influence between human and AI elements within information systems neither need be "exquisite" nor complicated. Rather, simplicity aids interpretation. On the other hand, in selecting analytical approaches it is important to note that not all applications are created equally, especially in complex information environments. In contrast to ecommerce and content delivery applications, User Interfaces (UI) designed for Analytical and Information Systems support synthesis. The market for these systems competes on effective user design. This means that while streaming service websites all look about the same, there are great differences between analytical UIs.

Wider variation between applications means that generalization of analytical approach can run aground "apples to oranges" problems very quickly. For example, web analytics for usage analysis and workflow analysis rely heavily on frequentist measures like 'click rate'. In content delivery applications and ecommerce applications, clicks correspond to selections of content. As such, 'click rate' has some face validity as a metric for user workload. In contrast, in analytical applications, click behavior can have different meaning or at least different weighting—GIS applications rely heavily on map-based UIs, which rely more on 'drag', 'mousedown', and scroll events at different rates to navigate maps. The meaning of these events is different to—what users' click on (the map) is arbitrary and doesn't indicate a selection.

Previous work illustrates that in analytical applications click-rate has less predictive value for workload, for example, than the pattern or distribution of clicks across different

features—how users integrate use of different elements with different information (Mariano et al 2015; Poore 2017). Additionally, the same research finds that models that incorporate frequentist features (e.g., the number of clicks) do not uniformly predict workload well across application with different design features. In summary, simplicity is important, but so is extensibility in analytical approaches—our aim is to provide analytical frameworks to the test community that accommodate a variety of input features in anticipation of wide variation between Analytical UIs.

Current Directions in Logging Human-AI Transactions

Our work addresses objectives in translating software system instrumentation into a dual-use technology—capitalizing on its ubiquitous use and corpus of enabling, open-source technology and providing capabilities to adapt it for use in HSI TEVV of human-AI interactions within Analytical and Information Systems.

Granular Behavioral Logging. To address granularity in capturing human interaction with software systems, we adopted a disaggregated, holistic approach to logging user behavior, which is to say, "log everything you can, filter what you know you don't need, and keep everything else". To accomplish this, we utilize a COTS open-source product: Apache Flagon UserALE.js. UserALE.js is a Javascript(.js) library for logging user behavior data in web applications. We selected it for its open-source posture (accessibility to the test community), permissible license (ALv2), its customizability, and features described below. It deploys as a .js 'script tag', a Node.js Package Manager (NPM) module, and as a Chrome/Mozilla browser plugin.

To meet HSI TEVV objectives for human-AI interactions, it is important to capture how human users interact with all elements of the user interface. Given these are designed for synthesis and a key TEVV objective is to understand the reciprocal influence between human and information space, it's essential to observe their interactions with all features (e.g., page elements) that manipulate or manage the information space. Additionally, owing to lessons learned, we also sample user behavior at a rate (e.g., every 500ms) and across the spectrum of user behavior types both at the page/document-level (e.g., 'click', 'mousedown', 'zoom', 'mouseover'), including behaviors that indicate users are attending outside of a given display (e.g., 'focus', 'blur'). UserALE.js provides this capability—by default it logs virtually all behavior on all elements on a page. Other products can be configured to do this as well (e.g., Matamo).

We also collect 'verbose' logs with comprehensive metadata. **Listing 1** provides a sample 'click' event logs from Apache Flagon UserALE.js, in its native Javascript Object Notation (json) format. This verbose log format provides a highly granular capture of user behavior with sufficient context to understand the behavior (e.g., 'click') the target element (named by HTML tag, e.g., 'test button'), as well as detail about the embedding of the target element in the Document Object Model (DOM), which adds additional information that aides in disambiguating behavior when target elements are similarly named.

In the course of our work, we developed additional capabilities to make UserALE.js maximally useful for the test community and increase granularity in behavioral data capture. This includes the ability to log from "iframes" with the UserALE.js browser plugin. Also, we have developed examples for how to customize logging through additional scripts (without UserALE.js source modification), these include, for example, examples for how to log XMLHttpRequests using UserALE.js' native custom log API. This kind of capability allows for capturing not only user behavior but also transactions between the client and server, as well as data sent back and forth. For many applications this would allow capturing some representation or record (e.g., URI, json blob) of new data sent to the client based on a user request (search, query, mouseover/'tooltip'). Similar methods can be used to capture 'fetch' or 'websocket' transactions as

Listing 1: Apache Flagon UserALE.js Behavioral Log

```
{
 target: 'button#test_button',
 path: [
    'button#test_button',
    'div.container',
    'body',
    'html',
    '#document',
    'Window'
],
pageUrl: '…',
pageTitle: '…',
pageReferrer: '…',
browser: '…',
clientTime: 1639544867213,
location: { x: 56, y: 210 },
scrnRes: { width: 1044, height: 439 },
type: 'click',
logType: 'custom',
userAction: true,
details: {},
userId: '…',
toolVersion: '…',
toolName: 'Apache UserALE.js Example',
useraleVersion: '2.2.0',
sessionID: 'session_1639544630593',
customLabel: '…'
}
```

well, and provide a means of associating interactive UI elements, with user behavior, and the underlying information space of Analytical and Information Systems.

Traceability in Instrumentation. Client-level instrumentation provides a high degree of granularity about user behavior and about how the system responds to user behavior, transactionally. However, the latter capabilities are limited to transactions in which the client (e.g., UI) initiates request or receives data. In larger, distributed analytical and information systems, components that 'serve' requests from clients are rarely the point of origin for new data or information that an analyst might request. AI services are frequently deployed as modules or microservices deeper into systems. Typically, other services route requests to these modules that initiate a request, or they operate asynchronously with the client and push data to other data services that client can query. Therefore, without other methods it is impossible to fully attribute data received by the client to specific elements, which poses inferential challenges for HSI TEVV.

Figure 3. How System Telemetry Instrumentation Works

System Telemetry instrumentation provides a wide toolset for adding 'silver-bullet' traceability for transactions between human and machine elements, especially when there are numerous intermediary transactions between the point of request (e.g., client) and the service that creates or processes requested information. **Figure 3** illustrates how System Telemetry works; lean instrumentation is deployed throughout the system (each module is language specific) that report a 'chain of custody' for requests as they are routed through the larger system. Each request has a unique 'traceId'. The "telemetry" data about that 'trace' is a series of log messages that declare which module reported participating in that trace and whether they were parent or child to another process in that trace. This process creates an unbroken chain of traceability between requests and responses.

Recognizing the value in System Telemetry for HSI TEVV, we are creating replicable examples for how to configure System Telemetry to report into Behavioral Usage Logs. This would allow us to add traceability to the granularity we are able to collect about human-AI transactions from the client. Specifically, we performed some experiments with Open Telemetry, and open-source (ALv2) COTS product for collecting and reporting system traces. They provide a 'user-interaction' module that does log user-behavior from the client that is associated with request to system processes. It doesn't provide comprehensive logging itself, however, it contains enough to cross-reference with Apache Flagon UserALE.js logs also generated from the client. This allows us to extract a 'traceId' from Open Telemetry logs and add them to UserALE.js logs, which allows us to relate user behaviors, especially requests for information with the source of that information (e.g., AI/ML modules). This approach is in keeping with our "dual-use" strategy and doesn't over-encumber the various roles and uses of instrumentation technology. It does provide significant equity in the technology for use in HSI testing and provides for access to the technology, methods, and best practices for utilizing System Telemetry to the larger test community.

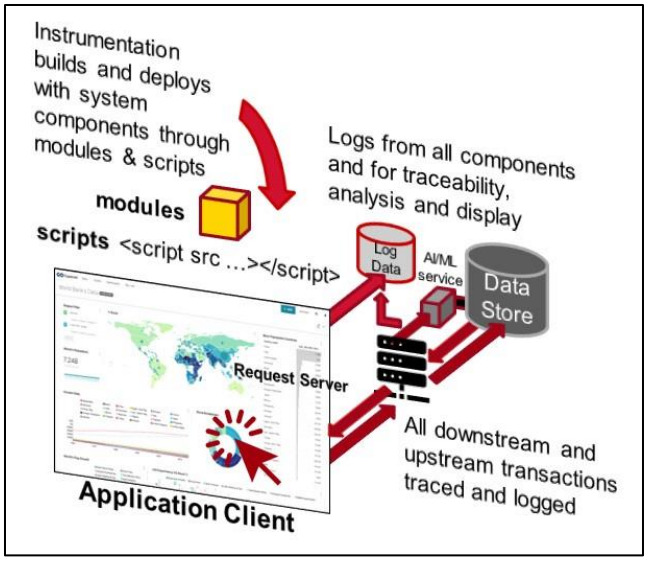

Figure 4. ARLIS' Open-Source Instrumentation Testbed

Another ARLIS development objective is to provide the larger test community with an open-source instrumentation testbed (**Figure 4**). The testbed itself is built from open-source components, including an open-source Analytical application (Apache Superset). Apache Superset provides significant latitude in designing Analytical dashboards constructed from underlying, modifiable Python utilities. This would allow the larger test community the ability to practice and prototype instrumentation methods using test domain-relevant data that may be imported to Apache Superset. To accomplish this, we are providing methods for injecting customizable UserALE.js and Open Telemetry instrumentation into Superset and providing accessible ways for the community to deploy the stack, and experiment with instrumentation data within customizable use-cases.

Current Directions in Pre-Processing Instrumentation Data

Understanding well that the biggest barrier to entry for utilizing Software System Instrumentation for complex behavioral analytics is data preparation and preprocessing, we are focusing our work on making it easier to for non-experts and novices to make use of this rich data. Specifically, our efforts center on two pre-processing pain-points. First, we are creating highly re-usable examples for incisive search, query, and filter operations to retrieve more manageable returns from data. Second, we are developing a Python package with exportable utilities and classes to support analysts as they *segment* instrumentation data—identifying and curating subsets of instrumentation data that correspond to specific users, software states, tasks, and workflows.

Search and Reference Utilities. To make search and reference operations for instrumentation data more accessible to the test community we adopted and improved upon precedents established by the Apache Flagon community. In this respect, we utilize Elasticsearch as a datastore to collect instrumentation data. Elasticsearch is built atop Lucene, which is specifically designed to facilitate document search. It supports a robust and easy to learn query language for terms pertaining to keys, values, and data stored in values. A widely used datastore, Elastic supports packages that make executing queries against it in a variety of analytical languages, such as Python (elasticsearch_dsl). As such, we have developed a range of re-usable examples (see **Listing 2**) for incisive search against UserALE.js, which can easily be modified to conform to other log data (e.g., Matamo).

Elasticsearch also supports a customizable dashboard application—Kibana—which allows users to explore log data in near-real time (see Figure 5). This provides an 'online' forensic capability that allow analysts and data scientists to understand how specific user events in client applications (e.g., analytical applications) result in specific patterns of logs. Using this capability, analysts and data scientists can establish how to describe the onsets (and exit criteria) of specific tasks in the application.

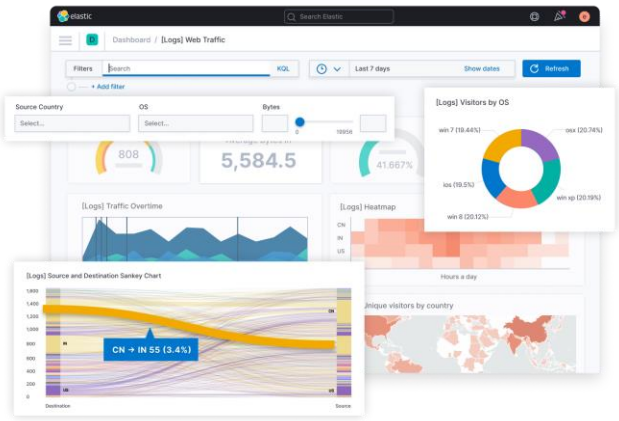

Figure 5. Elastic's Customizable Kibana Dashboard
© Elastic.co 2022

Segmentation Utilities. Perhaps the most challenging task in processing log data for analytical applications is identifying and curating *segments* of log feeds that correspond to specific tasks. This is also the most important task, as isolating logs that correspond to relevant user-driven operations provide the basis for inference within and across domain-relevant tasks. As above, even when onsets, entrance, and exit criteria for specific tasks are identified in instrumentation data, procedurally mapping them to segments and curating those segments amounts to non-trivial effort.

To address the challenge of creating and curating instrumentation data segments, we are developing a Python package that abstracts away many of the challenging operations required in easy-to-use classes and utilities. Once onsets, and entrance criteria can be identified our package will support the search and iterative capabilities to identify the relevant patterns of instrumentation data corresponding to those criteria utilizing efficient list and dictionary comprehensions. These utilities return a structured list of time/date pairs corresponding to the start/end times for those criteria. Then our package allows analysts to extract segments from

Listing 2: Example Workflow for Incisive UserALE.js Log Search and Reference in Elasticsearch

```
#define query
qLogType = Q("match", logType="raw") | Q("match", logType="custom")

#define filter
filterEvents = Q('bool', filter=[~Q('terms', type=['mouseover','wheel','keydown'])])

#chained search against Elastic
elk_search = UserALEsearch \
   .query(qLogType) \
   .query(qfilterEvents) \
```

these times as properties *objects* each with their own properties (**Listing 3**). Segment objects are then stored in a dictionary for search and reference. This provides two major benefits. First, this provides an efficient curation and forensic capability for analysts to evaluate each segment by a range of properties (e.g., length in time, number of logs) and apply additional operations to individual segments by name or by property (e.g., filter). Second, it will allow for a generative capability for creating new segments based on set logic (e.g., the union, intersection of two segments).

We believe our forthcoming package will significantly reduce barriers to entry for processing log data, aid the larger test community in building robust, re-usable instrumentation pre-processing pipelines, and enable the inferential capabilities necessary for understanding reciprocal influence between human and AI in complex analytical systems.

Current Directions in Generalized HSI Analytics for Instrumentation Data

Beyond understanding the basis of inference—how to make comparisons across various conditions (e.g., tasks, segments)—it is also imperative to consider how to operationalize influence in understand reciprocal influences between human and AI elements of larger systems.

The cleanest way to differentiate analytical approaches is whether the approach is descriptive of what users do or how they do it. For our purposes, the latter is more relevant. Analytical applications support users' synthesis of a wide swath of data and information. In that respect, frequentist information on what information they use based on what they click on is far less informative than how they integrate the information available to them to. Prior work supports this very intuitive point. Metrics summarizing integrative use outperform metrics of usage frequency (e.g., click-rate) by wide margins in predicting user accuracy on analytical tasks and user reports of Extraneous Cognitive Load (Mariano et al 2015; Poore 2017).

Another finding from the same prior work (Poore 2017) is that even integrative use metrics perform inconsistently across differently designed analytical applications if the models from which they are drawn are built from frequentist distributions. That prior work utilized "exquisite" Beta-Process Hidden Markov Models that characterized usage "states", where states were operationalized as distributions of click behavior across the element space (Mariano et al 2015). A powerful lesson learned from this work is in analytical applications (unlike ecommerce or content delivery applications), specific user-events (e.g., 'click') and their frequency carry different meaning for different applications.

To address prior lessons learned, we posit a structural approach to analyze and characterize how analytical and information systems are used. Specifically, we advocate for graph-based methods to evaluate how users sequentially chain their usage of different elements within analytical applications. This result is structural information (e.g., such as a directed graph) describing the temporal associations between different elements (**Figure 6**).

Analysis of these structures amounts to application of different graph analytics to one or more graphs. *Connectedness* and *Centrality* measures are analogous to the aforementioned 'integrated use' metrics. Combinations of metrics and weights may be useful for understanding the complexity of the graph. Graph methods also allow for comparison between one or more graphs in richer ways than are allowable by Markov states—*Edit Distances* provide a means to incisively characterize two graphs based on what structural changes are requisite to recreate one graph from another. Finally, graphs are also informative at different scales—Connectedness, for example is more dependent on the number

Listing 3: Example Workflow for UserALE.js Segmentation (create_segment)

```
#define segment based on toggle element value 'foo'
value = 'div.foo'

#create sequential start/stop time list from logs with value in DOM path of 'click' log
times = pairwiseStag([log['clientTime'] for log in sorted_data_paths_clicks.values() \
   if value in log['path']])

#create segments objects across data set based on definition above
Segments = segment.Segment.create_segment(clickData_dict, segment_names, times)
for d in Segments.values():
   print(d.segment_name, d.start_end_val, d.num_logs) #print segments with properties

segment1 ([start], [stop]) 17
segment1 ([start], [stop]) 23
segment1 ([start], [stop]) 35
```

edges than the number nodes. We believe that graph methods provide an analytical framework for characterizing how analytical applications are used than frequentist approaches.

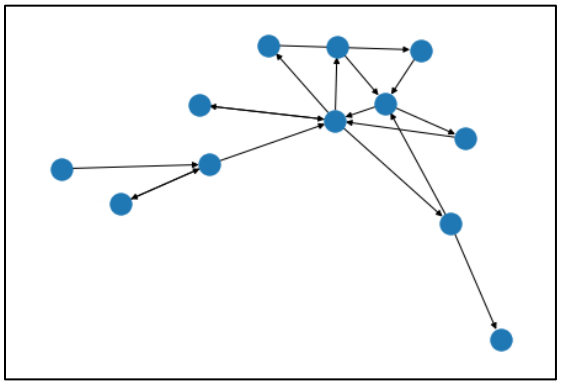

Figure 6. A Directed Graph Depicting Transitions between DOM Elements on 'Click' (Extracted from Segment)

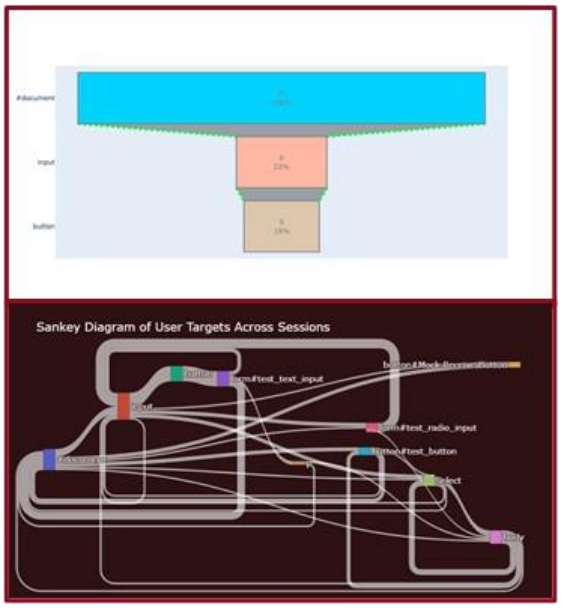

Figure 7. Workflow Verification Visualizations (Top: Funnel; Bottom: Sankey) Rely on Graph Methods

There are also pragmatic benefits to applying graph methods in this context. First, they can be used both descriptively and inferentially. A wide range of high-value, interactive visualizations for describing organic user workflows (e.g., Sankey; **Figure 7 (Bottom)**), and for testing hypothesis of specific workflows (e.g., Funnels; **Figure 7 (top)**) are built upon graph methods. Graphs are also accessible to a large audience with a variety of supportive analytical packages (e.g., NetworkX) and visualization libraries (e.g., Plotly) available for in major computing languages.

We advocate for graph methods as an effective basis for measurement within analytical and information systems. However, graphs alone are not a basis of inference for reciprocal influence. This is the importance of segmentation and the value of multi-method software instrumentation (behavioral and system logging). Using these methods together, we can segment behavioral instrumentation based on when the client receives new information AI/ML services, for examples, and trace these returns to user requests. By comparing graphs between segments, across segments—on metrics or structure—we can make inferences as to whether that information has a downstream influence on segments characterizing specific tasks. This creates natural experiments within log streams where a pattern of usage in tasks under different conditions (new data, or application states) has an influence on subsequent execution of the same tasks (**Figure 8**).

In our current work on instrumentation pre-processing and analytics, we are also reducing to practice utilities for structuring log data in dictionary formats (used in pre-processing) into formats that allow for easy ingest into graph methods. This includes easily creating and curating edge and node lists and doing so from extracted segments. We are also creating ready-made classes for creating graph visualizations (as above) and extracting graph methods. We expect to release (open-source) these analytical methods in the same package, side-by-side, with pre-processing methods.

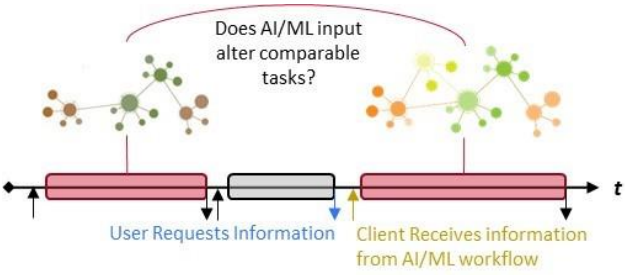

Figure 8. Illustration of the Basis of Inference for AI Influence on Human Workflows and Information Synthesis

Conclusions and Future Directions

In this paper we explored the core HSI TEVV objectives for evaluating human-AI/ML interactions within the context of analytical and information systems. We also remarked on why the chief objective—evaluating the influence of AI on human synthesis of information—is made more challenging in these systems as compared to autonomous control systems, for example. To address this objective, we argued that human system interaction need be captured behaviorally, such that transactions between human and AI elements of the larger system can be observed. To accomplish this, we further argued that granularity and traceability in measurement are both tantamount and elusive. Finally, we argued that software instrumentation provides a ubiquitous technology that can be adapted for use in HSI TEVV and provide a

measurement approach for gauging AI influence.

In developing a toolkit to bootstrap the HSI test community on instrumentation, pre-processing, and analytical methods we reached the following conclusions: First, *granularity* afforded by behavioral instrumentation provides us the information to create very specific segments based on specific tasks in the client. Database technologies like Elasticsearch make this easier with near-real-time dashboards and log indexing that support incisive query and filtering capabilities. Additional work by our team to create and curate segments as objects further reduces the work, providing the basis of a generative capability for creating additional segments based on the unions and intersections of existing segments. The value of segmentation cannot be oversold—creating granular segments provides the inferentially meaningful contrasts within the log stream. Without this capability we could not make inferences to AI influence.

Second, *traceability* afforded by system telemetry instrumentation allows us to attribute new software states and information sent to the client directly to user or system operations. Using existing technology (Open Telemetry) we are developing examples to guide integration between behavioral logging and system telemetry capabilities. Integration is lean and provides 'dual-use' benefit without adding unnecessary work or redundant functionality that would prompt adoption issues by developer communities.

Third, having learned from prior work, simple graph methods may provide a stronger and more intuitive analytical framework for creating methods for integrative use than 'exquisite' or 'bespoke' models based on frequentist distributions—classes of events and their frequencies mean different things in differently designed analytical applications. Using graph methods—metrics, contrasts—extracted from segments defined based on user and system data provides a pragmatic approach to measuring influence between human and AI elements of analytical and information systems.

Next Steps. We emphasized the importance of understanding *reciprocal* influence between humans and AI elements of larger information systems. Humans can exert influence on AI by affecting what data is uses and ultimately whether returns from AI are misused or misinterpreted. While our inferential framework provides a reasoned approach for gauging the influence of AI on human information synthesis, it doesn't fully address influence in the other direction. Likely, this dovetails with larger conversations about transparency in AI and require thoughtful consideration of how AI services should report on themselves. Should AI report on the confidence of its own output in a meaningful, and machine-readable way, it is likely that those reports (logs) can be cross-referenced with user behavior by way of co-associations with system telemetry logs.

We have already planned a series of human subjects research protocols to perform psychometric analysis for how metrics owing to our inferential framework for human-AI influence correlate with extant, established measures of related concepts (e.g., HCI Trust), and task performance measures. It is likely that this work involves more than simple correlation—subjective measures implicitly rely on vastly different timescales than analytics extracted from instrumentation data. As such, another consideration is how best aggregate output from our framework over sessions, for example, to enable *meaningful* correlations with subjective measures. This would avoid the need for interruptive protocols and prompts that could disrupt organic use.

Finally, we aim to continue our research and developmental efforts into methods for pre-processing and segmentation. One reason why the community hasn't fully adopted instrumentation (at this depth) into its practice is because of the skills-based "price of entry". Our commitment to example-based development, open-source releases, and permissible licensing will provide this accessibility.

Acknowledgement

This work supported by the United States Government under contract FA8702-15-D-0002. The view, opinions, and/or filings contained in this material are those of the author(s) and should not be construed as an official position, policy, or decision of the Government of the United States or Carnegie Mellon University or the Software Engineering Institute unless designated by other documentation.

References

Alonso, R., & Li, H. (2005). Model-guided information discovery for intelligence analysis. Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 269–270.

Lee, J. D., & See, K. A. (2004). Trust in automation: Designing for appropriate reliance. Human Factors, 46, 50-80.

Mariano, L. J., Poore, J. C., Krum, D. M., Schwartz, J. L., Coskren, W. D., & Jones, E. M. (2015). Modeling strategic use of human computer interfaces with novel hidden Markov models. Frontiers in Psychology, 6, 919.

Poore, J. (2017). FAST COGNITIVE AND TASK ORIENTED, ITERATIVE DATA DISPLAY (FACTOID). Charles Stark Draper Laboratory.

Wright, W., Schroh, D., Proulx, P., Skaburskis, A., & Cort, B. (2006). The Sandbox for analysis: concepts and methods. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 801–810.