[DISTRIBUTIONAL]

# Distributional's Testing Framework

# Intro

**Distributional has updated and continues to update our core functionality.**

**See docs.dbnl.com for the latest on our data pipeline, analytics workflow, and enterprise platform.**

For teams scaling AI applications in production, adaptive testing is critical for consistency and reliability. But testing these applications is not trivial. This is especially true for any LLM-based application, due to the nature of the embedding process, and subsequent recovery of text from embedding space, these apps require statistical analysis and testing.

Additionally, once an app is running in production, testing automation becomes necessary since teams can no longer manually test for all possible usages and analyze all possible responses. Even when ignoring the infeasibility of searching the space of all possible text, these teams are paid to ship things, not solely test them.

Distributional provides an adaptive testing platform that is uniquely designed to address these needs. Distributional's testing strategy is based on analyzing recent production usage to test for consistency of app behavior as a whole, while also providing mechanisms to both alert users to behavioral deviations and provide interpretable evidence for users to understand what has occurred.

In this paper, we describe Distributional's testing strategy in more detail, and how it can be applied for both live and synthetic production testing.

For additional resources, please visit:

- Distributional Docs
- Distributional Demos
- Distributional Blog

# Table Of Contents

# Strategy To Test For Consistent App Behavior In Production

At its core, Distributional's platform uses a testing strategy that embraces the idea that every time an AI app is used, both the prompt (input) and response (output) are random variables drawn from some distribution. The goal is to analyze the behavioral consistency of the app; for example, has the app responded to questions about the CEO of company X differently today than yesterday. The platform then surfaces notable deviations to users in an unsupervised way. By design, it does not pass judgment on the veracity of those responses; rather, Distributional gives users the ability to introspect and apply supervision by passing judgment based on their own expertise and their use cases.

To help discuss this testing strategy in terms of random variables, we introduce some notation. Capital letters represent a random variable, and lowercase letters represent a realization of that random variable or other deterministic quantity. (Note: For simplicity in this paper, we only discuss the input/output consistency from an app. Distributional's approach, in practice, also analyzes consistency across intermediate data generated by the app.)

- $I$ – the distribution of possible inputs (prompts) to the app
- $i$ – an observed set of prompts
- $O$ – the distribution of possible outputs (responses) from the app
- $o$ – an observed set of outputs
- $t$ – time span over which inputs and outputs are considered

Now, we also introduce some conditional notation to facilitate analysis in Distributional.

- $I \mid t_1$ – the possible inputs over time span $t_1$
- $I, O \mid t_1$ – the possible inputs and outputs over time span $t_1$
- $O \mid t_1, I$ – the possible outputs over time span $t_1$, given the inputs $I$
- $t_b$ – a baseline time period against which recent usage is compared

Distributional's core testing functionality studies the following hypothesis test:

$H_O$: $I, O \mid t_e$ and $I, O \mid t_b$ are the same distribution

$H_1$: They are not the same distribution

In general, $t_e$ is a recent time period and $t_b$ is a fixed time window from the past. For example, comparing usage from the past 24 hours to usage from last Monday.

Distributional then helps users understand whether they should reject $H_O$ by presenting relevant evidence of behavioral deviations based on logged app usage between $t_b$ and $t_e$. If the user finds this evidence compelling, notifications can be created to identify such behavior in the future.

This evidence can also be used to consider alternate, more targeted, null hypotheses. These could be:

$H_O$: $I \mid t_e$ and $I \mid t_b$ are the same distribution

which asks only whether the inputs to the app have changed or

$H_O$: $O \mid t_e$ and $I \mid t_b$ are the same distribution

which asks only whether the outputs have changed given the input distribution.

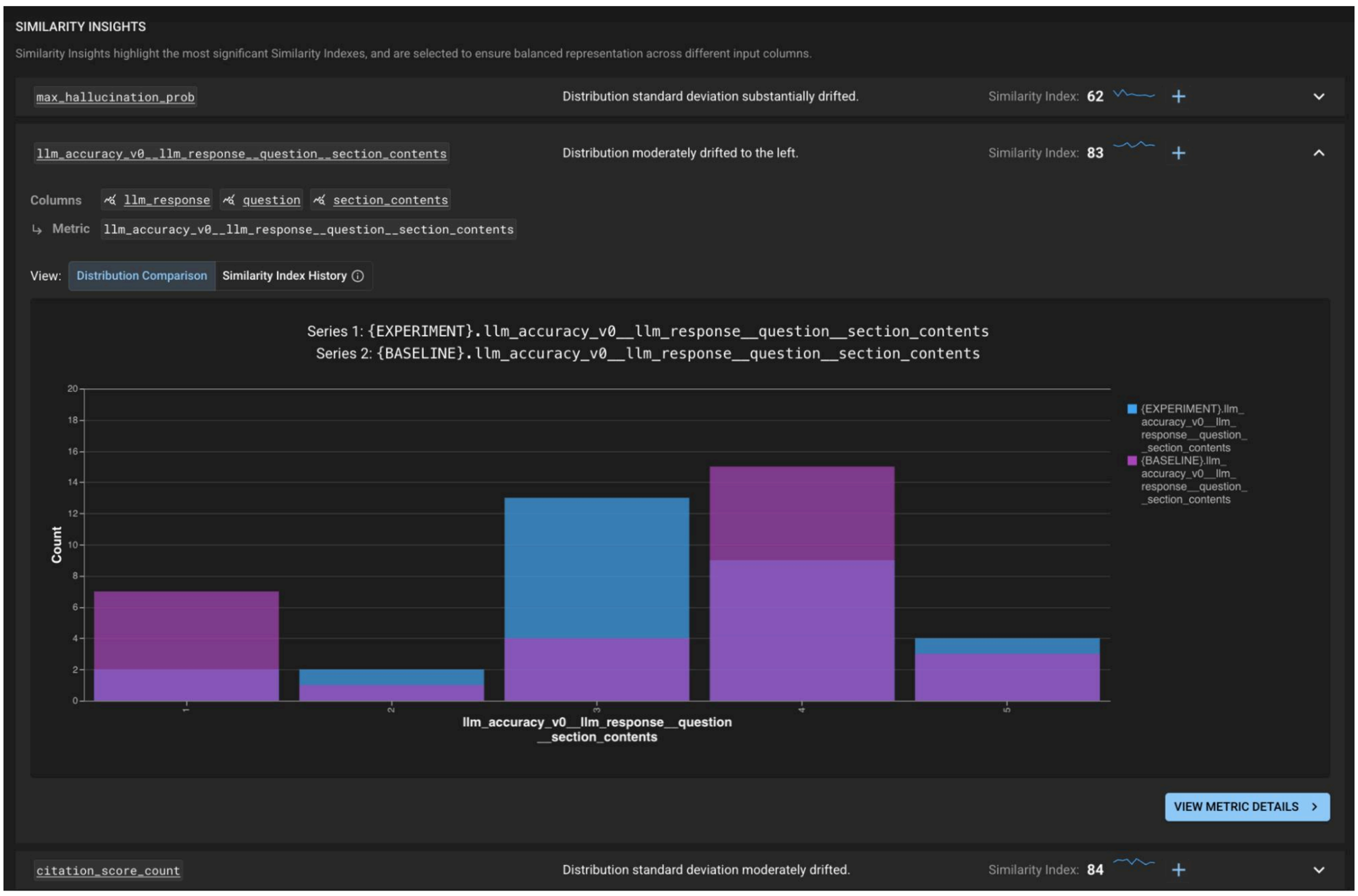


**FIGURE:** Evidence of a perceived deviation in behavior may be summarized with a statement such as "Distribution moderately drifted to the left," but users can dig deeper to interrogate that evidence and judge if this change in behavior is worrisome.

## BACKGROUND ON DISTRIBUTIONAL CONSISTENCY

Distributional consistency could be naturally analyzed under special circumstances. For example, the classic method of the *t*-test would be a logical strategy for considering consistency of normally *t*distributed data. If we were studying only the inputs $I \mid t$ and they consisted of only a single numerical value that was normally distributed, then we could analyze whether with $E[I \mid t_b] \neq E[I \mid t_e]$ with a *t*-test. But in the text-first world of generative AI, it is unlikely that such parametrized analysis will ever be sufficient.

A nonparametric analysis of distributional similarity has been well addressed for a single continuous or discrete random variable by tools such as the Kolmogorv-Smirnov statistic or Chi-squared statistic, respectively. Tools such as the Kullbeck Liebler (KL) Divergence provide a strategy to measure dissimilarity between random variables when the distribution of those random variables is known.

However, these tools alone are generally insufficient to analyze the consistency of observed app behavior since:

- the nature of text is neither numerical nor categorical;
- we desire to study multiple variables representing characteristic behavior of the text simultaneously; and
- we are unable to proactively sample data (for, e,g., the KL divergence) given fixed historical logs.

Nevertheless, Distributional does incorporate these quantities as facets for helping to define how severely an app's recent behavior has deviated from previously observed behavior. This is discussed in the next section.

## INCORPORATING INTERPRETABLE EVALUATION METRICS

Distributional is designed to empower users to ultimately make the judgment whether a significant or worrisome behavioral deviation has occurred. This means the platform must clearly provide understandable evidence of deviations to users.

To that end, the analysis of $H_O$ is powered by interpretable evaluation (eval) metrics. Distributional provides a set of built-in evals of different structures:

- Locally computed classical NLP quantities such as reading level
- LLM-as-judge style quantities leveraging the user's choice of model
- Hooks for users to create their own LLM-as-judge quantities for submission to Distributional
- RAG-specific quantities to help analyze the behavior of the retrieval process

Furthermore, any additional quantities that users already compute can be sent to Distributional to be incorporated into the analysis of $H_O$.

# Similarity Index For Studying The Hypothesis Tests

Distributional's testing strategy not only needs to analyze the behavioral consistency of an application, but results of this analysis also need to be understandable by users, regardless of the volume of logs, number of metrics, or complexity of the app. This is why Distributional created the Similarity Index (Sim Index) as an automatically calculated value—between 0 and 100—which defines the deviation between two time periods, or runs, and, in turn, serves as a proxy for the user to reject $H_O$. In effect, a lower Sim Index should make the user more likely to reject the null hypothesis. This can be done at the application level, for a grouping of metrics, or for a single metric or eval being logged. This strategy is how Distributional expands beyond the limitations of existing parametric and nonparametric methodologies.

## COMPUTING SIM INDEX

The core concept for computing a Sim Index starts with a single column of numerical or categorical data – this could be one of the evals that is computed on the text or a column of non-text data that the user has provided. We refer to this single column as $m_e$ and $m_b$ for the columns from the recent time period and baseline time period, respectively. Our strategy for 6 computing Sim Index on this column, denoted by $sim(m_e, m_b)$, is powered by a desire to produce *evidence* of dissimilarity between and $m_e$ and $m_b$. And then, through this evidence, the user can decide *memb*if these columns are, in fact, different in a way that is significant or relevant for their needs.

To surface such evidence, we assume that $H_O$ is true, i.e., $m_e$ and $m_b$ are drawn from the same distribution. We then pool results from both $m_e$ and $m_b$ to facilitate an independent and identically distributed (iid) bootstrapping process. This gives an initial sense of what should be happening if the two runs were actually drawn from the same distribution.

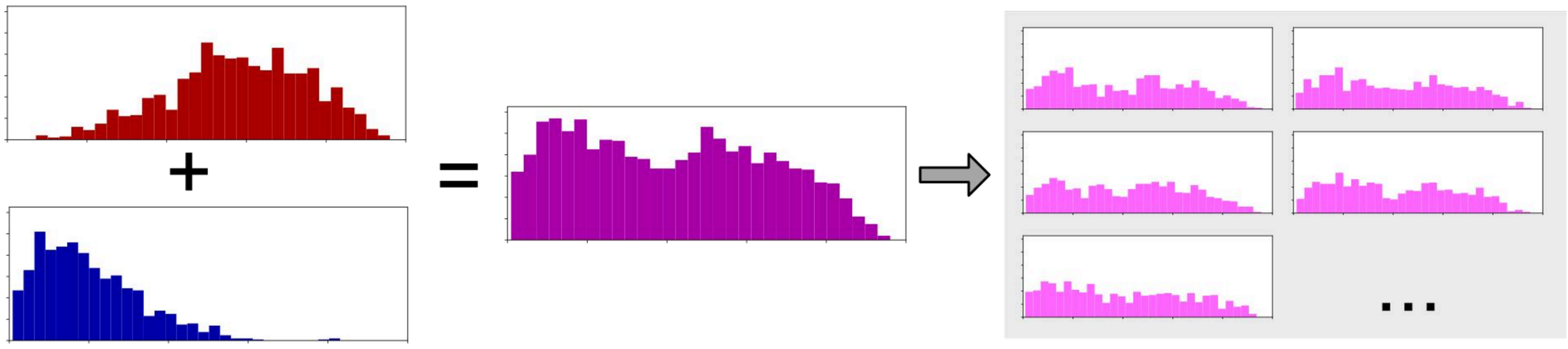

**FIGURE:** Data from today and previously observed data are pooled into a single population from which samples are drawn. These samples, in the box on the far right, provide a sense of what should be observed if today's data showed no deviation from previous data.

From each of these test pairs, common statistics for both $m_e$ and $m_b$ are computed. These include, for example, the 10th percentile, or the prevalence of the most common categories. This bootstrapping strategy has the benefit of "normalizing" and accounting for the scale of the quantities present to permit all subsequent analyses to take place on a fixed scale. These bootstrapped quantities also provide the desired evidence of behavioral deviation that is surfaced to the user.

The final Sim Index value for this column is computed through a weighted averaging of the difference of these common statistics as well as nonparametric measures such as the Mann-Whitney U statistic. Future versions of Sim Index may enable users to customize the weighting process to better align with their sense of behavioral deviation, e.g., more heavily weighting tail behavior rather than central tendency of the distribution.

## SIM INDEX FOR TEXT COLUMNS AND THE APP

A Sim Index can also be generated for a text column through combining the Sim Index values for all the quantities derived from that column, e.g., the sentiment or toxicity of a given column. That combination is primarily the minimum Sim Index value over the derived metrics. This conservative strategy ensures that any potentially troubling metric deviations are clearly surfaced to users. This also allows users to set thresholds on individual metrics or even statistics of those metrics so they can be notified of any deviations.

A Sim Index value is also generated at the app-level in the same fashion over all the columns in the app. This provides a helpful signal to users as to whether there's been deviations in behavior overall.

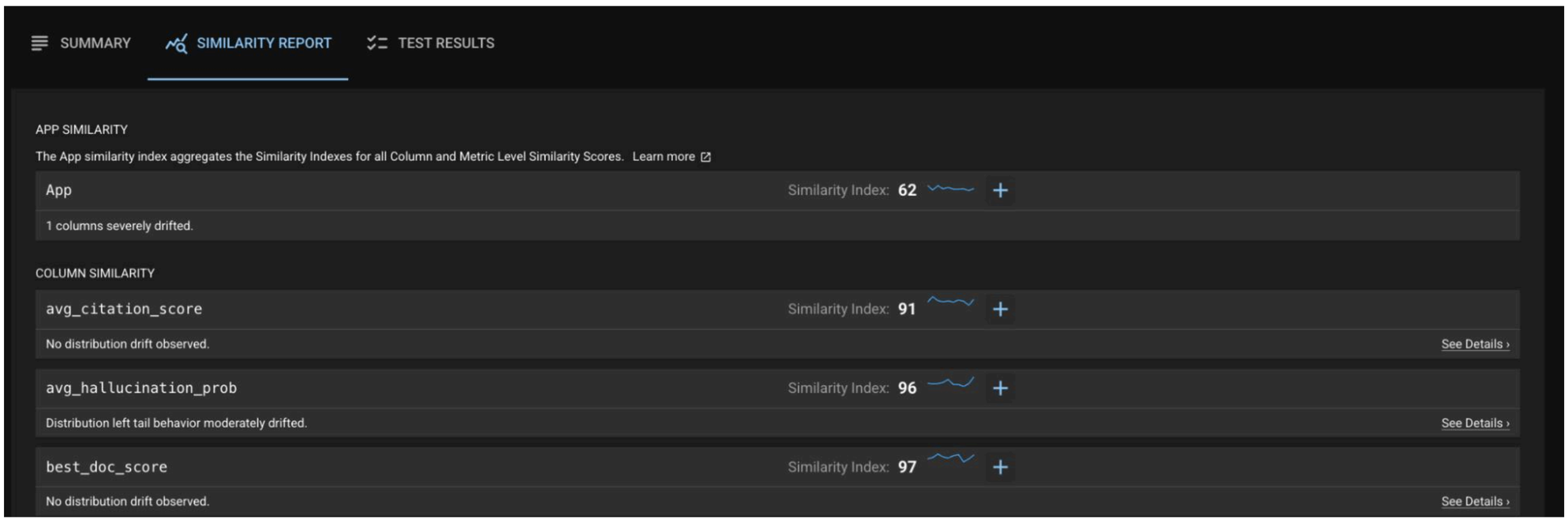


**FIGURE:** An app-level Similarity Index, built on column-level Similarity Index values and insights.

# Distributional for synthetic production testing

As described earlier, Distributional is designed to analyze recently observed app usage extracted from production logs to inform users of deviations in app behavior. Another investigative strategy with Distributional is to synthetically produce app usages using a fixed set of canonical inputs every day, such as a golden dataset.

By pushing a small number of consistent inputs through the app, the confounding analysis of "Which prompts did users submit?" can be skipped.

This is, in the parlance of experimental design, holding one parameter constant (the inputs) while allowing the other to vary (time) and analyzing only the effect of time on the app. One common use of this is in RAG-based knowledge management, where new documents are regularly added to the vector database as they arrive (such as 10Q financial documents). The app owner could run synthetic production testing in Distributional's platform to understand how the answers to consistent questions change as new documents arrive.

# Conclusion

Distributional's approach to AI testing provides a flexible and scalable framework that teams can get started with, regardless of the robustness of existing evals, and that can be applied to any AI application, regardless of complexity. Through the Sim Index and interpretable metrics, it enables teams with immediate signals on shifts in behaviors, and arms them with the relevant evidence to understand those deviations and pass judgment on whether they matter for their app. Ultimately, providing an automated workflow that AI product teams can use to continuously define, understand, and improve AI application behavior in production.

[DISTRIBUTIONAL]

# About Distributional

Distributional is building the modern enterprise platform for adaptive testing to make AI safe, secure and reliable. As the power of AI applications grows, so does the risk of harm. By taking a proactive, adaptive testing approach with Distributional, AI teams can deploy AI applications with more confidence and catch issues before they cause significant damage in production.

Learn more at distributional.com

**Follow us on:**

- LinkedIn
- YouTube