DEOXYS: A Causal Inference Engine for Unhealthy Node Mitigation in Large-scale Cloud Infrastructure

Chaoyun Zhang
Microsoft
China

Randolph Yao
Microsoft
USA

Si Qin
Microsoft
China

Ze Li
Microsoft
USA

Shekhar Agrawal
Microsoft
USA

Binit R. Mishra
Microsoft
USA

Tri Tran
Microsoft
USA

Minghua Ma
Microsoft
USA

Qingwei Lin
Microsoft
China

Murali Chintalapati
Microsoft
USA

Dongmei Zhang
Microsoft
China

ABSTRACT

The presence of unhealthy nodes in cloud infrastructure signals the potential failure of machines, which can significantly impact the availability and reliability of cloud services, resulting in negative customer experiences. Effectively addressing unhealthy node mitigation is therefore vital for sustaining cloud system performance. This paper introduces DEOXYS, a causal inference engine tailored to recommending mitigation actions for unhealthy node in cloud systems to minimize virtual machine downtime and interruptions during unhealthy events. It employs double machine learning combined with causal forest to produce precise and reliable mitigation recommendations based solely on limited observational data collected from the historical unhealthy events. To enhance the causal inference model, DEOXYS further incorporates a policy fallback mechanism based on model uncertainty and action overriding mechanisms to *(i)* improve the reliability of the system, and *(ii)* strike a good tradeoff between downtime reduction and resource utilization, thereby enhancing the overall system performance.

After deploying DEOXYS in a large-scale cloud infrastructure at Microsoft, our observations demonstrate that DEOXYS significantly reduces average VM downtime by 53% compared to a legacy policy, while leading to 49.5% lower VM interruption rate. This substantial improvement enhances the reliability and stability of cloud platforms, resulting in a seamless customer experience.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Causal reasoning and diagnostics**.

KEYWORDS

Causal inference, Failure mitigation, Reliability

ACM Reference Format:
Chaoyun Zhang, Randolph Yao, Si Qin, Ze Li, Shekhar Agrawal, Binit R. Mishra, Tri Tran, Minghua Ma, Qingwei Lin, Murali Chintalapati, and Dongmei Zhang. 2024. DEOXYS: A Causal Inference Engine for Unhealthy Node Mitigation in Large-scale Cloud Infrastructure. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3698038.3698534

1 INTRODUCTION

Large-scale cloud infrastructures are composed of millions of computing resources or nodes, each of which is monitored using a heartbeat signal to determine its health status. Occasionally, these signals may be lost due to a variety of reasons such as networking issues, hardware failures, or software bugs, resulting in the node entering an "unhealthy" state, which indicates a potential machine failure that can affect all virtual machines (VMs) on that node. The presence of unhealthy events can negatively impact customer experience, as their services and deployments may be disrupted until the machines are mitigated. Despite the existence of proactive systems [34] that predictively mitigate potential node failures before they occur, unhealthy events still frequently

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11.
https://doi.org/10.1145/3698038.3698534

occur in large-scale cloud infrastructures [21, 23], which can result in significant cloud outages and loss of millions of dollars [1].

When unhealthy events do happen, the highest priority is to choose an appropriate corrective action to the node, *e.g.*, Reboot, Redeploy, for mitigation that minimizes disruption to VMs deployed on that node and accelerates the recovery time [20], ensuring a smooth customer experience. However, choosing the most appropriate mitigation action for unhealthy events in a cloud environment is a complex task that presents several challenges. Ideally, immediate action should be taken when an unhealthy event occurs to minimize VM downtime. This however means that only limited information about the event is available, without knowing the root cause that led to the failure. Root cause analysis processes can take several hours or even days [39, 42, 61], and waiting for too long is not feasible in a cloud environment where prompt response is required. This incomplete information poses significant challenges to the mitigation engine, as it needs to make decisions based on limited signals. Furthermore, root causes of unhealthy events are often heterogeneous [13, 18, 28, 59, 65]. Taking different mitigation actions may therefore result in significant variance in the duration of VM downtime [34]. This heterogeneity further increases the reliability requirements of the system, as choosing the wrong mitigation action can potentially cause disruptive impacts to the cloud system.

An effective approach to address these challenges involves the integration of data-driven machine learning techniques [19, 71] into the mitigation engine [16]. However, many of these methods rely on online or reinforcement learning [34, 66], necessitating training through interaction with the production environment, which may result in performance regression during the exploration phase. Offline methods are also viable, although they demand a meticulous approach to mitigate potential confounders and ensure the acquisition of reliable experimental data. Otherwise, the predictions may be subject to data bias, resulting in less reliable recommendation [67]. A commonly used approach for experimentation to eliminate confounders is *action-level* A/B testing [34], where different mitigation actions can be compared and evaluated. However, conducting large-scale A/B testing for unhealthy node mitigation may not be feasible in a cloud environment, as it is typically randomized, regardless of the node status. This can lead to potential risks associated with taking incorrect mitigation actions on a large number of nodes, causing system performance regression, which can result in a poor customer experience and impact the reputation of the cloud provider.

This paper introduces Deoxys, an end-to-end causal inference mitigation engine engineered to bridge the aforementioned gap. Deoxys has been seamlessly integrated into Microsoft's expansive cloud computing infrastructure, and it's geared towards furnishing intelligent mitigation strategies in response to the occurrence of unhealthy events. One of its notable features is the capacity to be trained offline exclusively on observational data, obviating the need for action-level A/B testing and online learning. Once trained, it can be promptly deployed to recommend effective mitigation actions upon detecting unhealthy failures, thereby minimizing VM downtime and interruptions. This eliminates the need for the exploratory warm-up stages required by online learning methods, which enhances the reliability and stability of the overarching platform, optimizing the virtual machine experience for customers.

In the inference layer, Deoxys utilizes advanced causal inference techniques [52, 75], including double machine learning [30] and causal forest [63]. These methods allow for the generation of precise and dependable mitigation recommendations using constrained signals from the unhealthy node. Importantly, the models are trained exclusively on observational data, eliminating the necessity for expensive and disruptive randomized action-level experiments. In addition, Deoxys is capable of extracting simple logical "if-else" like policies from the data, which highlight the most important features driving its decision-making process. This interpretability is crucial in a cloud system for accountability, debugging, and troubleshooting purposes [10], allowing system operators and administrators to gain insights into the decision-making process of Deoxys.

Furthermore, Deoxys incorporates several dedicated system design elements to ensure its robustness and resource requirements are met within a cloud computing environment [2]. One such design feature is the inclusion of policy fallback, which enables Deoxys to automatically adjust its decision when it has low confidence. This approach helps to minimize the risk of erroneous decisions by the model, thereby enhancing its system reliability. Furthermore, Deoxys is designed to override an action if it determines that the gain in downtime reduction is minor compared to another action, but the tradeoff is a potential node resource saving. This allows Deoxys to strike a balance between minimizing downtime and optimizing capacity utilization. Additionally, if unhealthy events occur repeatedly at the same node within a short time period, another action override will be triggered to revert the Reboot to Redeploy, which prevents repeated failures from jeopardizing the system. All of these designs ensure that the overall system performance is optimized while maintaining robustness and capacity requirements.

We deployed Deoxys in a large-scale cloud infrastructure at Microsoft. Our findings revealed that Deoxys was able to effectively reduce the average VM downtime compared to the legacy policy for unhealthy nodes, while leading to substantially lower VM interruption rate. This has positioned

it as a critical component in the mitigation strategy of the cloud platform at Microsoft. Overall, our contributions are summarized as follows:

- We propose DEOXYS, a causal inference based unhealthy failure mitigation engine that recommends mitigation actions, without the need for large-scale, interruptive *action-level* A/B testing for data collection, and the exploratory warm-up stages required by online learning methods.
- We design a policy fallback and action override mechanisms in DEOXYS to improve the reliability of the system, while achieving a balance between downtime reduction and resource utilization.
- Offline comparisons using a high-fidelity simulator show that DEOXYS can achieve over 14% reduction in downtime compared to the state-of-the-art mitigation policy.
- We deploy DEOXYS as an unhealthy mitigation engine in a large-scale cloud infrastructure at Microsoft. Our observations show 53% downtime reduction, as well as 49.5% lower VM interruption rate compared to a legacy policy. These reductions potentially lead to better customer experience and huge business value.
- We extract logical policies from DEOXYS to gain insights from data and understand the features that drive its predictions, unveiling the decision-making process of DEOXYS.

To the best of our knowledge, DEOXYS represents the first deployment of an unhealthy mitigation engine based on causal inference in a large-scale cloud infrastructure.

2 BACKGROUND AND OVERVIEW

In a large-scale cloud system, a node is marked as "unhealthy" if a central controller loses communication with it for a specified time threshold. In such cases, the controller sends a diagnostic request to retrieve the status of the affected node. If the node is unable to self-recover, the controller must take mitigation action to restore the VMs operating on the node.

2.1 System & Objective

The aim of this research is to address the issue of smart failure mitigation in cloud systems when aforementioned unhealthy events occur. Specifically, we focus on the VM host environment, which represents a node in a cloud computing platform. This environment is composed of a complex stack of components, including guest OSes, guest agents, hypervisor, host OS, host agents, firmware, and hardware. The node is connected to various compute services, referred to as the controller, responsible for resource provisioning and management actions such as creating and destroying VMs.

The primary objective of this study seeks to develop advanced and intelligent policies that automatically identify the most effective mitigation action to minimize VM downtime and interruption when an unhealthy node is detected, so as to improve the reliability and availability of cloud systems [46, 62].

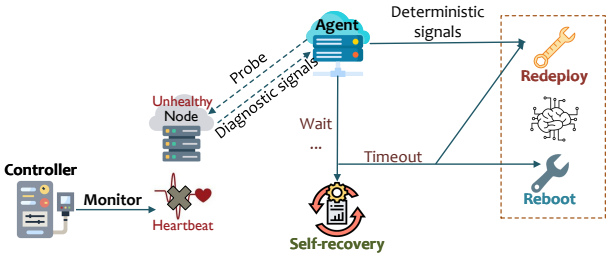

Figure 1: The workflows of unhealthy events and mitigation.

2.2 Unhealthy Event Workflow

In the context of production settings, modern cloud computing platform usually employs a controller to oversee and regulate the operational status of nodes. Nodes, which are the fundamental computational units in the cloud infrastructure, can exist in a variety of different states, the most common of which is the "ready" state wherein they are fully functional. On occasion, a node may transit into an "unhealthy" state, indicating its inability to respond to requests from the controller within a predetermined timeframe. This state may arise due to various factors, such as software and hardware malfunctions or network issues. In such scenarios, the controller may intervene to mitigate the unhealthy node by taking different actions, such as waiting for the node to self-recover or executing alternative mitigation measures. Nevertheless, such mitigations incur downtime for the node.

The specific set of procedures for managing unhealthy nodes are shown in Fig. 1. Firstly, the central controller loses communication with the heartbeat sender of a node, and if this communication loss persists for a specified duration, the node state is transitioned to unhealthy. Subsequently, if the node state remains unhealthy for a further period, the controller sends a signal to an agent, which triggers the unhealthy workflow in the system. The agent then performs several node diagnostic procedures, to further collect node data understand its unhealthy state. If the diagnostic results yield deterministic signals, such as an uncorrectable error, the agent immediately takes the node out of rotation and REDEPLOY the customer workload to a different healthy node. Alternatively, the agent waits for an unhealthy timeout, before taking any invasive action. Finally, if the node remains unhealthy after the timeout, the agent attempts to mitigate the situation by performing an in-place repair, such as a node REBOOT, or triggering a REDEPLOY by transiting all VMs to

other healthy nodes. Choosing an appropriate mitigation action is critical because it can lead to different VMs downtime. This forms the foundation objective of this research.

We note that, grey failures, where a node experiences performance degradation but does not fully fail, typically do not trigger the unhealthy node mitigation process because the node's heartbeat remains active. Therefore, such failures are handled by mechanisms outside of Deoxys.

2.3 Diagnostic Signals

The proposed Deoxys and the controller rely on diagnostic signals to probe the system status of unhealthy nodes. These signals provide important insights that can reduce the observability gap [27] and facilitate effective decision-making.

- **VM Information**: Number of VMs, types of VMs, session types, and whether important workloads are running on the node. These are used to understand the current workload and resource utilization of the node to decide on an appropriate mitigation action.
- **Network Information**: Network connectivity status of the node, used to determine if network issues are contributing to the unhealthy node status to make appropriate mitigation decisions.
- **Error Code**: Helps diagnose the root cause of the issue. Though sometimes missing, it can provide valuable diagnostic information in certain scenarios for more effective mitigation.
- **Unhealthy Repeat Times Signal**: Frequency of the node becoming unhealthy in the past. It is an indicator of impending failure, which helps in proactive mitigation actions.
- **Uncorrectable Error Tag Signal**: Indicates whether the node can be corrected. It is to decide whether to take corrective actions or to leave the node for human investigation.

Overall, these diagnostic signals provide a rich set of features that help understand the state of the unhealthy node and inform appropriate mitigation decisions.

2.4 Mitigation Actions & Legacy Policy

In the event that a node becomes unhealthy, it is incumbent upon the central controller to take one of two actions to restore the VMs operating on the affected node. Unless the node undergoes a self-recovery, the central controller is responsible for ensuring the proper functioning of the affected VMs. These actions include:

(1) **Reboot** action constitutes a mechanism that facilitates the preservation of VM state in the event of a host Operating System (OS) reboot [9]. Fundamentally, the process entails reloading the host OS kernel into memory while maintaining the persistence of the VM and device state to the reloaded kernel. The reboot is then executed with the loaded kernel, and the persisted state is retained, while the rest of the state in the previous kernel is discarded. Upon the commencement of the reloaded kernel, the preserved state is restored, albeit with a momentary interruption. In Reboot, VMs are restored at the same node without requiring additional resources.
(2) **Redeploy** action is an approach that ensures the seamless transfer of a running VM from one host to another using a live migration [25] or service healing [24] engine, with minimal disruptions. The process involves migrating the VM's memory, processor, and virtual device state. The engine facilitates iterative copying of the VM's memory pages while preserving a dirty page set for the VM on the source host. After the VM has been halted, the engine synchronizes the dirty state with the target host and resumes the VM on the target host. Notably, various reasons may lead to the failure of Redeploy, *e.g.*, shortage of node resource. In the case of Redeploy, all VMs will encounter a brief pause similar to a Reboot. Note that the Redeploy action entails the migration of VMs to a different node, which requires additional resources.

The current study is centered on the scenario whereby the controller loses the heartbeat for a specified time interval, prompting the need to select either the Reboot or Redeploy as a mitigation action.

Legacy Policy The management of unhealthy nodes previously relies on a legacy mitigation policy that employs heuristic rules for making mitigation decisions based on diagnostic signals. It predicates on an offline data collection approach centered on the success probability of the repair action. It entails waiting for a predefined, heuristic duration derived from data accumulated over a two-year period preceding the policy modification. Should the machine fail to spontaneously recover within this time frame, a Reboot was initiated, followed by an extended wait before determining whether to Redeploy the VMs to an alternate machine. Notably, the heuristic waiting period was solely determined by the hardware type and limited diagnostic signals, which does not take into account any of the workload characteristics or other environmental attributes. This simple policy may overlook important factors and criteria between mitigation action and VM downtime. In this study, we compare the performance of Deoxys with this heuristic legacy policy, as it is the only policy deployed in production.

2.5 Key Performance Indicators

The key performance indicators (KPIs) of the unhealthy mitigation task are mainly evaluated from two perspectives: *(i)* Average VM Downtime (AVD); and *(ii)* Annual Interruption Rate (AIR) [34]. For a single unhealthy mitigation request on a machine node $\mathcal{N}$, AVD is defined as the average downtime

by all VMs on that node:

$$\textbf{AVD} = \frac{\sum_{\text{VM}\in\mathcal{N}} Y_{\text{VM}}}{\text{VM count}}. \quad (1)$$

This metric represents the primary objective of DEOXYS and is the key optimized metric for our system. Additionally, we also consider the AIR, which is defined as:

$$\textbf{AIR} = \frac{I_{\mathcal{T}}}{L_{\mathcal{T}}} \times 365 \times 100. \quad (2)$$

In this equation, $\mathcal{T}$ represents the duration of the measurement interval, expressed in days. For the purposes of this paper, we set $\mathcal{T}$ to 4 months, corresponding to the full experimental period. The variable $I_{\mathcal{T}}$ denotes the number of VM interruptions that occurred within the interval $\mathcal{T}$, while $L_{\mathcal{T}}$ represents the total VM-lifetime within the same period, measured in days. The constant 365 is used to scale the rate to an annualized figure, and the factor of 100 adjusts the result to represent the number of interruptions per 100 VMs annually.

In the context of this study, a VM interruption refers to a forced reset of a VM, initiated by automated mitigation actions. The AIR metric specifically captures the frequency of these unhealthy events per 100 VMs in a year and is independent of their duration. A lower AIR value corresponds to fewer interruptions. We choose this metric for several reasons. First, even brief interruptions can severely degrade user experience, particularly for latency-sensitive applications such as online gaming. Additionally, based on feedback from our customers, frequent interruptions are perceived as more disruptive and frustrating than a single, longer period of downtime. AIR is thus a meaningful measure of service reliability and user experience in cloud environments, as it effectively quantifies the frequency of such interruptions.

Note that due to the dynamic relationship between the mitigation policy and the node state, the mitigation actions can significantly impact the AIR. Inappropriate actions may lead to new unhealthy events and VM interruptions occurring in quick succession. Although the AIR is not a direct optimized objective of DEOXYS, we closely monitor this metric to ensure that the policy derived from DEOXYS does not result in significant regressions on AIR.

2.6 Challenges

In addressing the unhealthy event mitigation problem, we acknowledge several challenges that need to be overcome.

- **Challenge 1: Observational Data Contamination**: The data available for model training is observational, as the legacy policy, on which the mitigation engine relies, does not conduct randomized experiments for each action. Observational data are contaminated by confounding variables, making it challenging to discern the true causation between mitigation action and downtime.
- **Challenge 2: Incomplete Information**: The lack of complete information regarding the unhealthy node and its root causes poses a significant challenge, particularly when utilizing data-driven methods. The uncertainty stemming from incomplete information must be considered in the mitigation design.
- **Challenge 3: Resource Limitations**: The REDEPLOY action, which requires migrating all VMs from an unhealthy node to a healthy one, may be limited by the system's available resources. This poses a practical challenge in ensuring that the system can perform the REDEPLOY action without overwhelming the system's resources or causing additional disruptions.
- **Challenge 4: Mitigation Engine Mistakes**: The potential for mistakes by the mitigation engine is a significant concern, as a single wrong decision could potentially lead to disastrous consequences. It is crucial to incorporate a protection mechanism to minimize the risk of erroneous decisions by the engine.

Addressing these challenges in the design of DEOXYS is essential to ensure the reliability of the system.

3 CAUSAL MODELING

The stringent requirements for high reliability and interpretability in unhealthy mitigation render causal inference highly suitable, a perspective often overlooked by traditional systems. Specifically, for causal models: *(i)* they learn an unbiased policy from observational data generated by the legacy policy, obviating the need for costly action-level A/B testing for data collection; *(ii)* they can be trained offline, without the need of warm-up stage on online environment; *(iii)* they provide high explainability by extracting if-else-like policies (see Sec. 7). DEOXYS thus leverages causal inference as the foundation for constructing its mitigation policy.

To first investigate the relationship between node downtime and mitigation actions, we construct a causal model to explicitly analyze unhealthy events for optimizing the mitigation decision-making process. This process involves:

(1) Constructing a causal graph that delineates the interrelationships between various system variables.
(2) Formulating a causal machine learning model based on the causal graph, which serves to rectify data biases inherent in legacy policies and estimate the causal effect of mitigation actions on VM downtime.
(3) Predicting the potential downtime difference when a mitigation action is selected using the causal model. We then base our mitigation policy on the predicted values.

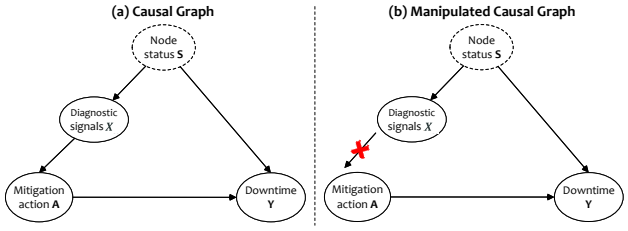

Figure 2: The causal graph (a) and the manipulated causal graph (b) of the unhealthy node mitigation.

We detail these processes in the following subsections.

3.1 A Causal View of Unhealthy Events

Based on the unhealthy node workflow shown in Fig. 1, we can derive an abstract graphical causal model that represents the causal relationships depicted in the left plot of Fig. 2(a).

The causal graph constructed comprises four sets of variables, namely S, $\mathbf{X}$, A, and Y:

- S captures the overall status of an unhealthy node, such as the software and hardware conditions, and the root cause of the issue *etc*. However, the complete status S is not fully observable. S can be partially characterized by the observable diagnostic signals $\mathbf{X}$.
- $\mathbf{X}$ comprises N diagnostic signals (described in Sec. 2.3) that are obtained by probing the status of the unhealthy node, *i.e.*, $\mathbf{X} = \{X_1, X_2, \cdots, X_N\}$. These signals are fully observable, enabling us to design appropriate mitigation strategies. Nonetheless, $\mathbf{X}$ provides less information compared to S, thereby leaving a greater degree of uncertainty.
- A denotes the mitigation action taken after an unhealthy event, where $A \in \{0, 1\}$. The value $A = 0$ corresponds to the Reboot, while $A = 1$ denotes the Redeploy.
- Y quantifies the duration of VM downtime that occurs subsequent to the mitigation action. It is calculated as the average value of downtime across all VMs in a node.

In addition, edges in the causal graph depict the causal relationships between the variables. Specifically,

- $S \rightarrow \mathbf{X}$: The status of the node S influences the set of diagnostic signals $\mathbf{X}$. Given S is not fully observable, $\mathbf{X}$ becomes the only signals that can be used to profile the node state and formulate a proper mitigation policy.
- $\mathbf{X} \rightarrow A$: In the legacy policy, the mitigation action A is determined by the diagnostic signals $\mathbf{X}$, based on heuristic rules and subject to randomness, as described in Sec. 2.4.
- $(S, A) \rightarrow Y$: The duration of VM downtime Y is influenced by both the node status S and the mitigation action A. This is because the recovery of VMs follows different workflows for different mitigation actions, each of which is associated with a different processing time.

The graph depicts that variables S and $\mathbf{X}$ act as confounding factors [52], as they simultaneously influence both the mitigation action A and the downtime Y. This arises due to the limitations of collecting only observational data from the legacy policy.

Problem: Confounding factors have the potential to distort the relationship of $P(Y|A)$ from the true causation between mitigation action (A) and VM downtime (Y), leading to imprecise and biased estimates of their true effect. This can compromise decision-making, as illustrated by Simpson's paradox [64], where these factors can make interventions appear more or less effective than they truly are by influencing both the mitigation action and the VM downtime. To prove, we transform the correlation probability $P(Y|A)$ as follows:

$$\begin{aligned} P(Y|A) &\overset{(1)}{=} \sum_{\mathbf{X}} P(Y, \mathbf{X}|A) \\ &\overset{(2)}{=} \sum_{\mathbf{X}} P(Y|A, \mathbf{X}) P(\mathbf{X}|A) \\ &\overset{(3)}{\propto} \sum_{\mathbf{X}} P(Y|A, \mathbf{X}) P(A|\mathbf{X}) P(\mathbf{X}) \end{aligned} \tag{3}$$

The concept of marginal distribution underpins Line (1). The subsequent Lines (2) and (3) are derived from the Bayesian rule. In particular, the term $P(Y|A, \mathbf{X})P(A|\mathbf{X})$ in Line (3) is influenced by the observed data. Specifically, the selection of a particular action for a given $\mathbf{X}$ in the dataset $P(A|\mathbf{X})P(\mathbf{X})$ leads to an increase in the value of $P(Y|A)$ and consequently introduces bias in the estimation.

While one remedy approach can be *action-level* A/B testing, where actions are taken independently of diagnostic signals $\mathbf{X}$, this is impractical as it is prohibitively expensive and could disrupt the system. Instead, causal inference provides a method to deconfound the mitigation policy without the need for action-level A/B testing, ensuring more accurate assessments of mitigation effectiveness and addressing the observational data contamination in **Challenge 1**.

3.2 Deconfounding the Mitigation Policy

To accurately estimate the causal effect of a mitigation action (A) on VM downtime (Y), we need to account for confounding factors. Our goal is to calculate $P(Y|do(A))$ instead of $P(Y|A)$. $P(Y|do(A))$ represents the causal effect of actively intervening on A to see its direct impact on Y and helps us overcome biases introduced by confounders that could distort the observed relationships.

3.2.1 $P(Y|do(A))$ vs. $P(Y|A)$. The probability distribution $P(Y|do(A))$ differs from the $P(Y|A)$, as follows:

- $P(Y|do(A))$: This represents the scenario where we actively manipulate A to see its direct effect on Y. It isolates

the effect of A on Y by removing the influence of confounders $\mathbf{X}$.
- $P(Y|A)$: This represents the observed association between A and Y, which can be affected by confounders and does not imply causation.

To estimate $P(Y|do(A))$, we need to block the path from the confounders ($\mathbf{X}$) to A and reconstruct the causal graph as shown in Fig. 2 (b), to ensure that the relationship between A and Y is purely causal. This can be achieved by backdoor adjustment.

3.2.2 Backdoor Adjustment. We use the backdoor adjustment [11, 26, 67] technique to estimate $P(Y|do(A))$ using observational data. According to the backdoor criterion [43]:

- **Blocking Backdoor Paths:** $\mathbf{X}$ blocks all backdoor paths from A to Y.
- **No Descendants:** $\mathbf{X}$ does not include any descendants of A.

With these criteria met in Fig. 2, we can translate the causal estimand $P(Y|do(A))$ into a statistical estimand that can be calculated using observational data:

$$P(Y|do(A)) = \sum_{\mathbf{X}} P(Y|A, \mathbf{X})P(\mathbf{X}). \tag{4}$$

Note that the unobservable variable S is eliminated from the equation after the adjustment, as blocking $\mathbf{X} \rightarrow A$ is sufficient to satisfy the backdoor criterion. This equation allows us to emulate aspects of experimental design within observational studies, providing a more precise estimation of the causal effect between A and Y, even in the presence of confounding variables. We apply the backdoor adjustment technique to our dataset to estimate the Individual Treatment Effect (ITE). This helps us predict the difference in downtime when implementing one mitigation action over another, informing the design of our policy.

3.3 Individual Treatment Effect Estimation

Using the backdoor adjustment equation, we can train machine learning models to calculate the Individual Treatment Effect (ITE) [57, 75] for each unhealthy event. The ITE, denoted as τ_i, is defined as $\tau_i := P(Y_i|do(A_i = 1)) - P(Y_i|do(A_i = 0))$. This represents the potential downtime difference when choosing one mitigation action over another:

- **Positive τ_i ($\tau_i > 0$):** Indicates that choosing action $A = 1$ (e.g., Redeploy) results in longer downtime for event i, so we should choose action $A = 0$ (e.g., Reboot).
- **Negative τ_i ($\tau_i < 0$):** Indicates that choosing action $A = 0$ (e.g., Reboot) results in longer downtime for event i, so we should choose action $A = 1$ (e.g., Redeploy).

By selecting the optimal mitigation action based on the sign and value of τ_i, we can design an effective mitigation policy to minimize overall VM downtime.

Let $Y_i(0)$ and $Y_i(1)$ represent the potential outcomes of VM downtime when taking actions $A = 0$ and $A = 1$ for unhealthy request i, *i.e.*, $P(Y_i|do(A_i = 1))$ and $P(Y_i|do(A_i = 0))$ respectively. Using the backdoor adjustment, the ITE for request i is:

$$\begin{aligned}
\tau_i &:= E[Y_i(1)|\mathbf{X}_i] - E[Y_i(0)|\mathbf{X}_i], \\
&:= \sum_{\mathbf{X}_i} E[Y_i(1)|A_i = 1, \mathbf{X}_i]P(\mathbf{X}_i) \\
&\quad - \sum_{\mathbf{X}_i} E[Y_i(0)|A_i = 1, \mathbf{X}_i]P(\mathbf{X}_i), \\
&:= E[Y_i(1)|A_i = 1, \mathbf{X}_i] - E[Y_i(0)|A_i = 1, \mathbf{X}_i].
\end{aligned} \tag{5}$$

By estimating the downtime difference for each individual unhealthy request, we can optimize the mitigation policy to minimize the downtime duration for each event, consequently reducing the overall VM downtime for the entire system.

3.4 Double Machine Learning

We use Double Machine Learning (DML) to design our mitigation policy for unhealthy requests. DML is a powerful method for estimating ITEs [15, 30, 35, 36], especially when dealing with high-dimensional data where traditional statistical methods fall short. In our context, we have a large number of diagnostic signals ($\mathbf{X}$) and a complex relationship between the mitigation action (A) and the VM downtime (Y). DML helps us accurately estimate the causal effect of A on Y by controlling for these confounders. While DML does not directly implement the backdoor adjustment as traditionally defined in causal inference, it aligns with the principles of backdoor adjustment by aiming to control for confounders to accurately estimate causal effects. DML operates in two main stages, namely *(i)* prediction stage and *(ii)* residuals stage, as shown in Fig. 3.

Prediction Stage: We first build two models: One to predict the VM downtime (Y) using the diagnostic signals ($\mathbf{X}$). Another to predict the mitigation action (A) using the same diagnostic signals. Mathematically, we represent these models as:

$$\tilde{Y} = f(\mathbf{X}, W_1), \quad \tilde{A} = g(\mathbf{X}, W_2). \tag{6}$$

Here, f and g are our predictive models, parameterized by W_1 and W_2 respectively. $\tilde{Y}$ and $\tilde{A}$ are the predicted values.

Residuals Stage: Next, we use the predictions from the first stage to perform a residuals-on-residuals regression. This isolates the effect of the mitigation action (A) on the

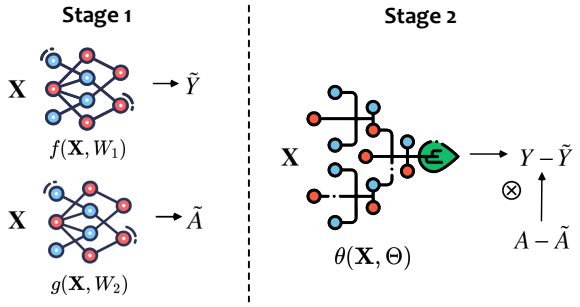

Figure 3: The two-stage DML framework.

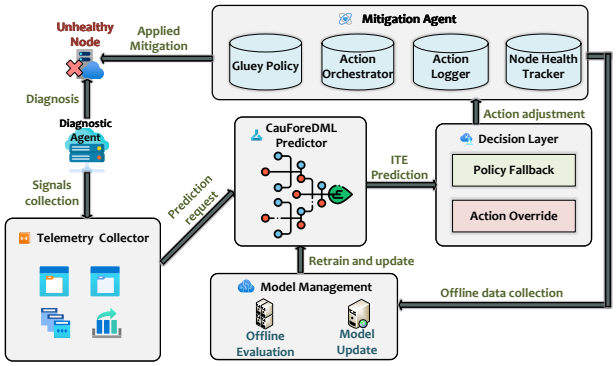

Figure 4: The overall Deoxys system architecture.

VM downtime (Y):

$$Y - \tilde{Y} = \theta(\mathbf{X}, \Theta) \cdot (A - \tilde{A}) + \epsilon. \quad (7)$$

Here, θ is our final model, parameterized by Θ, and ϵ is a noise term. We optimize Θ to minimize the error in this regression:

$$\tilde{\Theta} = \arg\min_{\Theta} \mathbb{E}\left[\left((Y - \tilde{Y}) - \theta(\mathbf{X}, \Theta) \cdot (A - \tilde{A})\right)^2\right]. \quad (8)$$

The key idea behind DML is to remove the bias introduced by confounders. By predicting Y and A first and then focusing on the residuals, we effectively isolate the true impact of A on Y. This two-stage process ensures that our estimates are not skewed by other factors [47]. To further reduce bias, DML uses a technique called cross-fitting. This involves splitting the data into different subsets and training the models on these subsets separately. Think of it as getting multiple opinions before making a decision, which helps average out biases and leads to more reliable estimates.

Once trained, our final model $\theta(\mathbf{X}, \Theta)$ can directly estimate the ITEs using the diagnostic signals. We discard the initial models f and g during inference, making the process efficient and scalable. In our study, we use multi-layer perceptrons for f and g, and a causal forest [50, 63] for θ, which will be detailed next.

3.5 Causal Forest

The causal forest is an extension of the random forest model [7] tailored to estimating the causal effects. The causal forest is an advanced version of the random forest model, specifically designed to estimate causal effects. While a random forest partitions data based only on input variables, a causal forest also considers the treatment variable (A) when making these partitions.

In a causal forest, each tree is split based on both the input variables ($\mathbf{X}$) and the treatment variable (A). This creates subgroups within each leaf of the tree, where the distribution of A is balanced using propensity scores. Propensity scores [29, 54] help ensure that the treatment and control groups are comparable, effectively controlling for confounders. By balancing the treatment variable within each subgroup, the causal forest can more accurately estimate the treatment effect of A on the outcome (Y). This method integrates causal inference principles with machine learning to capture complex relationships in the data while reducing bias.

One of the key features of the causal forest is its ability to provide confidence intervals for its predictions. This is achieved using a technique called Bootstrap-of-Little-Bags [4, 17]. Confidence intervals are crucial when dealing with incomplete diagnostic information, as they give a measure of the uncertainty in the predictions. This feature helps address **Challenge 2** by providing more reliable decision-making under uncertainty.

We combine the causal forest with DML to create a more accurate and robust model for estimating ITEs. We call this combined approach **`CauForeDML`**. Compared to other ITE estimators like the meta-learner [33], Deoxys employs `CauForeDML`, which uses two-stage estimation and cross-fitting to reduce bias from model regularization and overfitting. Additionally, by incorporating causal forests, `CauForeDML` provides confidence intervals for its predictions, aiding in decision-making under uncertainty. More details on this will be provided in Sec. 4.2.

4 DEOXYS DESIGN AND IMPLEMENTATION

The foundation of Deoxys rests on the core task of predicting ITE using the `CauForeDML` described in Sec. 3. Various components have been devised to guarantee the secure and dependable deployment of the `CauForeDML`. We provide an overview of the overall architecture of Deoxys in Sec. 4.1, and delve into the details next.

4.1 System in a Nutshell

The proposed Deoxys architecture is illustrated in Fig. 4, and it comprises five distinct components that collectively support the mitigation selection policy. These components include a telemetry collector, a `CauForeDML` predictor, a decision layer, a model management module, and a mitigation engine. To recap the process depicted in Fig. 1, when an unhealthy event occurs, a diagnostic request is generated and sent to an agent, which then collects diagnostic signals and forwards them to the telemetry collector. If the unhealthy event exceeds a timeout threshold, Deoxys is activated, and the `CauForeDML` (Sec. 3) predicts the ITE between different mitigation actions. The predicted ITE values are then used as input to the decision layer, which combines them and adjust its predictions using policy fallback (Sec. 4.2) and action override mechanisms (Sec. 4.3). The selected mitigation action is passed to the mitigation agent (Sec. 4.4), to applies the action to the unhealthy node and logs and tracks the outcome of the request. The model management module controls the model update process (Sec. 4.5), tracking the online performance and updating the `CauForeDML` as needed.

4.2 Policy Fallback

In certain scenarios, the `CauForeDML` model may generate predictions with low confidence scores, indicating that the available diagnostic signals may not be sufficient for the model to make a confident recommendation. This situation may arise when the input data is incomplete or ambiguous, compromising the model's ability to accurately predict the outcome. The confidence scores are determined by the gap between the lower bound and upper bound of the prediction, with a larger gap indicating lower confidence. To address this issue, we have designed a fallback mechanism to the legacy mitigation policy in cases where the predicted ITE exhibits significant variance, *i.e.*, $\tau_u = \tau_{upper} - \tau_{lower} \geq \varepsilon$, and the predicted ITE is close to zero, *i.e.*, $|\tau| \leq \xi$, where ε is the confidence threshold and $\xi \approx 0$. This indicates that the model is uncertain about its prediction and which action will lead to shorter downtime. In such cases, the legacy handcrafted policy can provide better performance and greater explainability. The fallback mechanism enhances the reliability of decision-making by incorporating the legacy policy when the data-driven model's predictions may be less reliable, thereby serving as a safeguard to ensure system reliability, addressing **Challenge 2** in Sec. 2.6.

4.3 Action Override

In addition, predicted ITE from the `CauForeDML` may provide insights primarily focused on optimizing VM downtime, potentially overlooking other critical aspects such as node resource requirements and repetitive failures. Relying solely on this for decision-making may result in suboptimal outcomes. To mitigate this, we design two action override mechanisms, which allow for reverting decisions made by the `CauForeDML` to improve system reliability. These mechanisms also enable the Deoxys to integrate human expertise into the decision-making process, providing a safeguard against potential errors made by the `CauForeDML`.

Changing Redeploy to Reboot. The decision of whether to perform a Reboot or a Redeploy action may also depends on the current capacity of the cloud system. Recall that the Redeploy action requires the migration of all VMs to different nodes, which may be constrained by scarce node resources in the system. In such cases, the controller may opt for a Reboot action to save more resource. To address this issue, we propose an action override rule in Deoxys that overrides the Redeploy action before querying the capacity of the system, based on the quantity of the estimated ITE τ_i. The absolute value of τ_i represents the potential downtime savings if one action is taken over the other. We set a override threshold for Redeploy as ϖ, such that if $\tau_i < 0$ and $|\tau_i| < \varpi$, we can override the Redeploy action with a Reboot, resulting in the saving of more nodes required by Redeploy, while only increasing subtle downtime duration. The parameter ϖ is set by minimizing a cost function that weighs the trade-off between downtime and resource usage, incorporating factors like revenue and user experience. [1]

By leveraging this mechanism, Deoxys is able to achieve a more favorable tradeoff between VM downtime and node resource, addressing the **Challenge 3** outlined in Sec. 2.6.

Changing Reboot to Redeploy. In addition, given the data-driven nature of Deoxys, it is susceptible to making errors, which can potentially result in undesirable system behaviors and a negative experience for consumers. For instance, in cases where hardware failures are the root cause of an unhealthy event, following Reboot recommendations may not necessarily resolve the underlying issue. Persistently recommending Reboot in such situations may lead to repeated VM interruptions within a short timeframe, as the underlying problem remains unresolved.

To address this challenge, another action override mechanism has been implemented in Deoxys. This mechanism involves flagging a node as having an issue if unhealthy events occur repeatedly on the same node within a short period of time. In such cases, a Redeploy action may be enforced, regardless of Deoxys's recommendation, and the node is marked as unallocatable for further investigation by human operators. This proactive approach helps to reduce the occurrence of unhealthy events and improves customer

[1] Due to the anonymization policies, we cannot disclose the specific formula of the cost function.

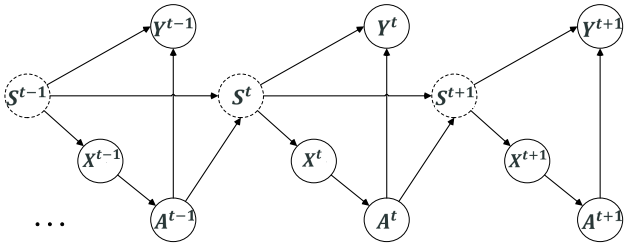

Figure 5: The dynamic causal graph of an unhealthy event.

satisfaction, thus significantly enhancing the overall reliability of the system. This reduces the risk when Deoxys makes a mistake, as discussed in **Challenge 4** in Sec. 2.6.

4.4 Mitigation Agent

After receiving the action adjusted in the decision layer, Deoxys employs several components that work together as a mitigation agent to repair unhealthy nodes. These components are the Sticky Policy, Action Orchestrator, Action Logger, and Node Health Tracker, which communicate with each other and external services.

Sticky Policy [44] The efficacy of Deoxys is evaluated through *policy-level* A/B testing compared against the legacy policy. This is opposed to the *action-level* A/B testing between mitigation actions. Assigning a policy to a node or unhealthy request is a critical aspect of A/B testing. To ensure consistency throughout the testing process, it is necessary to assign a policy that remains the same for the duration of the experiment. In the classic A/B testing setting, units are assigned randomly under the assumption that they are independent and identically distributed (i.i.d). However, this assumption can be problematic when dealing with nodes in a cloud system, as the status of a node at time t, denoted by S^t, is not necessarily independent of previous behaviors or mitigation actions. Consecutive unhealthy events on the same node can be highly influenced by previous node statuses and behaviors. For example, if a node experiences hardware failures, a simple Reboot action may not resolve the issue, resulting in new unhealthy events occurring in quick succession. If different policies are assigned to the same node during an experiment, the i.i.d assumption can be violated. We show this from a causal view in Fig. 5. The new time-dependent causal links between historical node statuses S^{t-1}, mitigation actions A^{t-1}, and the current node status S^t, create a Markovian process [14, 22, 69, 70] and violate the i.i.d assumption.

To address this problem, we propose the use of a "sticky policy". For each node, the group is determined by hashing the node ID and policy group name. If a node is assigned to policy P for an experiment, it will continue to use policy P for subsequent unhealthy requests. This ensures that the policy remains consistent for each node throughout the A/B testing process, and the presence of time-dependent causation does not violate the backdoor criterion, as $\mathbf{X}^t$ continues to effectively block all backdoor paths originating from A^t and leading to Y^t in Fig. 5. This ensures the continued validity and reliability of our causal modeling approach.

Action Orchestrator The action orchestrator plays a critical role in executing the action plan developed during the policy tree walk session. It follows instructions from the decision layer and sticky policy, making API calls to the relevant compute managers to implement actions. As actions can be implemented by different managers, the orchestrator executes them asynchronously to prevent blocking.

Action Logger Effective logging is important for data analysis, offline data collection, and evaluating different mitigation policies. In Deoxys, the logging format must record not only the chosen action but also the associated predicted ITE and corresponding confidence intervals. The mitigation agent logs the unhealthy timestamp, action timestamp, experiment name, model type, model name, model version, predicted ITE, confidence interval, chosen action, chosen action parameters, any triggers for action override or policy fallback, and the reasons for those triggers.

Node Health Tracker The node health tracker is responsible for monitoring node and VM health during the mitigation process. It tracks whether a VM is interrupted during mitigation and the time of its recovery. Additionally, it detects unsolvable node failures, such as hardware issues, and marks the node as unallocatable instead of waiting for resource brokers to allocate resources.

4.5 Online Model Update

The cloud environment is dynamic and continuously evolving due to factors like system updates and changes in customer workloads. To keep the Deoxys system effective, we have designed a model update process involving several key steps. Initially, we collect recent unhealthy events from the past month to update the training dataset, which is used to retrain the DML model at the core of our system. This update ensures that the model incorporates the latest data patterns and trends.

Once the model is retrained, we conduct offline evaluations to assess its performance. This evaluation involves testing the model on a newer dataset that was not used during training, ensuring its ability to generalize effectively to new data. Subsequently, upon confirming that the new model outperforms the existing one, we deploy it online, replacing the old model. This process occurs on a regular weekly basis.

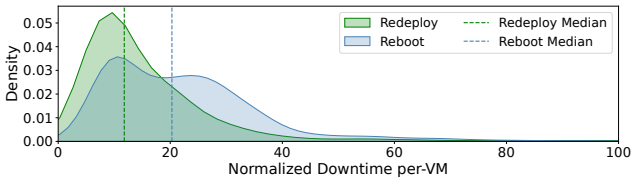

Figure 6: The normalized average VM down time distributions for both REBOOT and REDEPLOY in the offline dataset.

By adhering to this process, the DEOXYS system consistently utilizes the most accurate and up-to-date model to determine the optimal mitigation action for each unhealthy event. This proactive approach to model maintenance addresses data drift and minimizes the risk of using outdated models, thereby maintaining high performance levels over time.

5 OFFLINE EVALUATION

The `CauForeDML` is trained offline using a dataset from the real production environment. The model is constructed using the widely recognized causal inference tools `dowhy` [58] and `EconML` [5].

5.1 Compared Methods

Given that the ground-truth optimal mitigation action is unknown, we have devised a simulator to emulate the behaviors of unhealthy events within the cloud infrastructure at Microsoft. This simulator serves as a tool to assess the effectiveness of DEOXYS and enables offline testing of various mitigation policies. It is constructed based on historical unhealthy events, encompassing various types of failures that have been verified by engineers to maximize fidelity. For comparison, we assess the performance of *(i)* a **random** algorithm, *(ii)* the **legacy policy** currently deployed online, *(iii)* **Narya** [34], which employs bandit algorithms to explore and optimize mitigation policies in online environments, and *(iv)* **Nenya** [66], which utilizes hierarchical reinforcement learning to develop an optimal mitigation policy in real-time. Both Narya and Nenya necessitate learning within an online production environment, which includes a cold-start period [6, 73] before reaching convergence. Within the scope of causal inference methods, we also compare the accuracy of estimation among various approaches, as detailed in Sec. 5.4.

5.2 Offline Dataset Analysis

We collected an offline dataset from the production environment spanning a duration of one month to train the initial `CauForeDML` model. This dataset comprises a total of 20,218 unhealthy events, with each sample containing diagnostic signals $\mathbf{X}$, the mitigation action A, and the average VM downtime Y. The data is generated based on the legacy policy and is purely observational in nature.

Table 1: The offline comparison of mitigation policy in 10,000 simulated unhealthy events.

Policy	Random	Nerya	Nenya	**DEOXYS**
Downtime	12.1	6.4	5.6	**4.8**
Convergence Events	0	136	1,350	0

To gain insights from the offline dataset that used to train the causal model, we conducted a brief analysis. In Fig. 6, we present the distributions of average VM downtime (*i.e.*, $P(Y|A=0)$ and $P(Y|A=1)$) as well as their median values for both REBOOT and REDEPLOY. Note that due to the company policy, we hide the actual time and normalize as time unit. Observe that in general, opting for REDEPLOY results in shorter VM downtime compared to REBOOT. However, this is subject to the bias discussed in Sec. 3.2, and consistently selecting the REDEPLOY action may not necessarily be optimal. Moreover, the substantial overlap between $P(Y|A=0)$ and $P(Y|A=1)$ signifies the existence of a "gray area" and a blend in VM downtime concerning the mitigation action.

We also observe an intriguing phenomenon where nodes undergoing the REBOOT action exhibit a bi-modal distribution in downtime. This can be attributed to differing recovery outcomes post-REBOOT. In the first mode, some VMs recover rapidly, resulting in minimal downtime. In contrast, the second mode occurs when certain VMs fail to recover immediately, necessitating additional mitigation steps, which prolongs the downtime. This bi-modal distribution reflects the variability in reboot outcomes and recovery processes.

5.3 Offline Performance Comparison

In Table 1, we present an offline performance comparison of various mitigation policies using 10,000 simulated unhealthy events. The time units have been normalized and anonymized for consistency. Our results indicate that DEOXYS achieves the lowest downtime of 4.8 units, which is a 14.3% improvement over the best baseline, Nenya. This positions DEOXYS as a more reliable mitigation solution for the online environment. In contrast, Narya, which employs bandit algorithms but disregards the diagnostic signals as contextual information, performs worse. These findings underscore the superiority of data-driven policies over handcrafted ones, as it is challenging to design rules that generalize effectively across all cases.

We also compare the number of unhealthy events required for convergence across all methods. Our proposed method, DEOXYS, is trained on an offline dataset, thereby necessitating no exploration in the online environment and consistently

Table 2: Performance evaluation on different causal models.

Model	Loss ψ
LinearDML	12.1
SparseLinearDML	8.3
KernelDML	6.4
NonParamDML	5.6
CauForeDML	**4.8**

delivering robust performance from the outset. In contrast, Narya and Nenya require 136 and 1,350 events, respectively, to achieve convergence. This translates to days or even weeks of exploration before these methods can deliver stable performance if deployed online, potentially adversely affecting user experience. This underscores the advantage of Deoxys, which eliminates the cold start by training offline.

5.4 Comparison with Causal Baselines

In addition, we compare several variations of DML models with the CauForeDML employed in Deoxys on the same offline test set. These baselines include *(i)* **LinearDML**: This baseline uses an unregularized linear model as the final estimator [15]. *(ii)* **SparseLinearDML**: This baseline employs an $l1$-regularized linear model as the final estimator, aiming for sparsity [8]. *(iii)* **KernelDML**: This baseline utilizes a kernel function to better capture non-linearity in the effect model [48]. *(iv)* **NonParamDML**: This baseline makes no assumptions on the form of the effect model and uses XGBoost [12] as the final estimator [32].

We utilize the error function ψ averaged on the test set as a quantitative measure to assess the accuracy of ITE estimation on a test set, defined in Eq.(8). A lower value of ψ indicates higher accuracy in estimating ITE. The results are presented in Table 2. Notably, CauForeDML, integrated into Deoxys, achieves the best performance, demonstrated by the lowest ψ score. Compared to the other baselines, CauForeDML achieves up to a 15% reduction in final-stage loss, underscoring the superiority of combining DML and causal forest for ITE estimation. This highlights CauForeDML's effectiveness as a precise and robust ITE estimator in real-world production environments.

5.5 Offline Counterfactual Analysis

The capability of causal inference empowers us to conduct a counterfactual analysis [45] on the offline dataset using the CauForeDML. This analysis involves estimating the duration of downtime that would have occurred if a different action A' had been taken to represent a counterfactual scenario, *i.e.*, $A' = 1 - A$. Such a "what-if" analysis allows us to revisit the legacy policy in a postmortem manner and shed light on

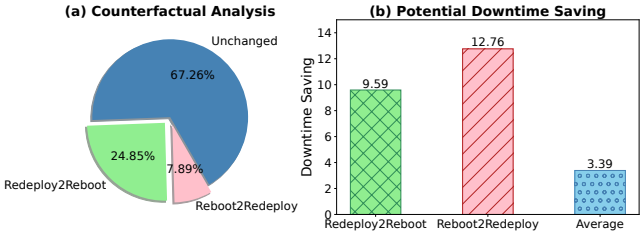

Figure 7: A counterfactual analysis on the offline dataset generated by the legacy policy.

the potential downtime savings that could be achieved by transitioning to the causal policy employed in Deoxys. It is important to note that these analysis results are based solely on offline data collected under the legacy policy and are not associated with the action override mechanisms discussed in Section 4.3.

Fig. 7 presents the results of the counterfactual analysis conducted on the legacy policy, assuming a different action is taken. In subplot (a), we present the estimated proportion of unhealthy requests that could result in shorter downtime if a different action A' had been taken. Notably, we observe that the action chosen by the legacy policy, which corresponds to 67.26% of samples, aligns with the causal policy. However, the CauForeDML predicts that a different action should be taken for 33.74% of unhealthy requests, resulting in reduced downtime compared to the original action. Interestingly, the CauForeDML estimator suggests that 24.85% of the mitigations that were originally assigned to the Redeploy should switch to the Reboot, which is more than three times the proportion of mitigations that should switch from the Reboot to Redeploy. This underscores the potential benefit of leveraging causal inference in making policy decisions.

We further analyze the predicted downtime savings for each switching group. In the group where the Redeploy should be taken instead of the Reboot, the average downtime saving is estimated to be 9.59 units, whereas in the group where the Reboot should be taken instead of the Redeploy, the average downtime saving is projected to be 12.76 units. This suggests a potential average reduction in downtime of 3.39 units if the legacy policy is replaced with the CauForeDML, and demonstrates the potential benefits of leveraging CauForeDML for guiding mitigation.

6 ONLINE ASSESSMENT IN PRODUCTION

We have deployed Deoxys as a core unhealthy mitigation engine in the cloud infrastructure of Microsoft. To compare its performance against the legacy policy, we have conducted *policy-level* A/B testing in production with comprehensive evaluations. Our A/B testing aims to answer the following research questions (RQs):

- **RQ1:** What is the effectiveness of Deoxys in reducing VM downtime and affecting AIR?
- **RQ2:** What impact does Deoxys have on other system metrics, such as blackout and capacity?
- **RQ3:** How can we leverage model uncertainty to design a reliable policy fallback?
- **RQ4:** Under what conditions should we override Redeploy with Reboot, and what are its effects?
- **RQ5:** Under what conditions should we override Reboot with Redeploy, and what are its effects?

6.1 Deployment Scale and Experiment Method

We conducted *policy-level* A/B testing on 23 production regions in the cloud infrastructure of Microsoft, spanning a duration of over 2 months, to compare the performance of Deoxys and the legacy policy[2]. This accounted for approximately one-third of the total nodes in production. Furthermore, for the sake of evaluation, we have also randomly designated two small sets of nodes to consistently apply either Redeploy or Reboot actions. Due to their limited number, the impact on the overall cloud infrastructure is expected to be negligible. The experiment commenced on February 25th, 2023, and ended on June 25th, 2023. This coverage encompassed a significant portion of nodes in the cloud, as well as a substantial number of unhealthy requests. Nodes were randomly hashed into either the Deoxys group or the legacy policy group with equal probability using the sticky policy, as described in Sec. 4.4. Note that only unhealthy requests that lacked deterministic signals for diagnosis and exceeded the unhealthy timeout were included in the A/B testing. This accounts for 55% of the total unhealthy requests. We have opted not to compare our Deoxys with the frameworks proposed in [34] and [66] in production, as they demand substantial enhancements to our existing infrastructure in order to facilitate the requisite real-time interaction.

6.2 Downtime & Interruption Reduction (RQ1)

The KPIs utilized to evaluate the effectiveness of Deoxys are the average VM downtime (AVD) and annual interruption rate (AIR), as described in Sec. 2.5. Table 3 presents the improvement achieved by Deoxys compared to the legacy policy in terms of percentiles, and average values of AVD and AIR. In total, 845 and 802 unhealthy events trigger the legacy policy and Deoxys during the A/B testing, with 5,138 and 4,587 VM were impacted respectively. Furthermore, the policies of always applying Reboot and Redeploy actions were triggered by a limited number of unhealthy events (98 and 106 cases). Due to the small quantity, these events are not expected to exert a significant adverse impact.

[2]The A/B testing is conducted to compare policies rather than actions, which is distinct from the action-level A/B testing introduced in Sec. 1 and 3.2. Policy-level A/B testing is acceptable and low-cost to the infrastructure.

Table 3: The improvement of AVD and AIR achieved by Deoxys compared to other policies in the A/B testing.

Policy	Sample	VM count	AVD gain				Avg. air gain
			P50	P75	P90	Avg.	
Legacy	845	5,138	49%	45%	44%	53%	49.5%
Reboot	98	522	65%	62%	61%	63%	54.3%
Redeploy	106	559	25%	23%	23%	24%	12.6%
Deoxys	802	4,587	-	-	-	-	-

Table 4: Blackout duration improved by Deoxys.

Blackout Duration	Avg.	P50	P75	P90	P99
Improvement	3.6%	5.4%	3.4%	0.5%	2.6%

Notably, Deoxys exhibits lower AVD values across all percentiles as well as the average value, demonstrating significant improvement in comparison to the legacy policy[3]. Specifically, Deoxys achieves reductions of 49%, 45%, and 44% in AVD for P50, P75, and P90, and a remarkable 53% reduction in average AVD. Moreover, although AIR is not a direct optimization objective, Deoxys still outperforms the legacy policy by a substantial margin of 49.5%. This highlights the hidden correlation between AVD and AIR, with Deoxys effectively addressing both VM downtime and interruption rate. It is crucial to note that the legacy policy serves as a strong baseline, having been designed by experienced engineers and employed in the Microsoft cloud infrastructure for years. Given the extensive scale of the cloud infrastructure and the frequent occurrence of unhealthy events, the significant gains achieved by Deoxys positively impact a large number of customers.

Likewise, in comparison to the consistently applied Reboot and Redeploy policies, our Deoxys consistently demonstrates superior performance across all metrics, including AVD and AIR. This underscores the importance of customizing mitigation strategies for specific unhealthy events, as a one-size-fits-all approach is suboptimal. Notably, while the Redeploy policy outperforms the legacy approach, it necessitates a significant allocation of additional node resources to accommodate the redeployed VMs, in contrast to rebooting a node, which retains the recovered VMs on the same nodes. This resource-intensive nature of Redeploy is the principal reason for its non-deployment in real-time online scenarios.

Table 5: Unallocatable metrics improved by Deoxys.

Unallocatable Metric	Ratio	Avg.	P50	P75	P90
Improvement	0%	12.0%	0%	12.8%	17.9%

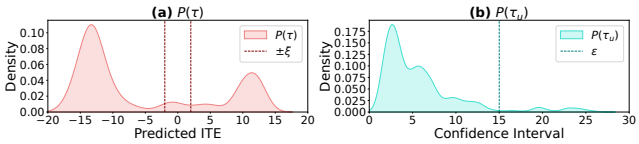

Figure 8: The distribution of the (a) estimated ITE τ and (b) its confidence interval τ_u.

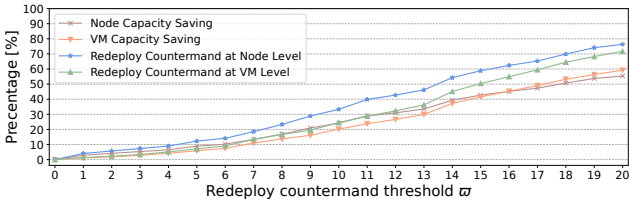

Figure 9: Potential node/VM saving as well as their override ratio w.r.t the Redeploy override threshold ϖ.

6.3 Node Interruption Assessment (RQ2)

The mitigation policy introduces interruptions to unhealthy nodes, such as *(i)* **Blackout**, which can cause blips and pauses to the deployed VM, leading to performance regressions such as jitters and slowdowns, and *(ii)* temporally **Unallocatable**, which occurs during the duration and process of mitigation action and results in the node being out of capacity and unable to be allocated with new VMs. Note that a shorter blackout/unallocatable duration and a smaller unallocatable rate indicate better system reliability. We evaluate these metrics to provide additional insight into the impact of Deoxys, for a more comprehensive assessment.

We present the reduction improvement achieved by Deoxys in terms of the blackout duration, unallocatable rate, and duration over the legacy policy in Tables 4 and 5, respectively. Our results show that Deoxys outperforms the legacy policy, with improvements in terms of average and all percentile values for blackout duration. This means that customers will experience less performance regression, resulting in a seamless user experience. Furthermore, as shown in Table 5, while both policies exhibit similar unallocatable rates, employing Deoxys can shorten the average unallocatable duration by 12%. This is crucial for the system's capacity, as a shorter unallocatable duration means that the overall cluster can accommodate more VMs in a given period.

6.4 Policy Fallback on Model Uncertainty (RQ3)

As discussed in Sec. 4.2, we have designed a callback mechanism based on the predicted ITE τ and its confidence interval τ_u. The purpose of this fallback mechanism is to enhance the reliability of the system in cases where the model feels uncertain or when the diagnostic signals are insufficient to make a decision. In Fig. 8, we present the distribution of ITE estimation τ and its confidence interval τ_u. We observe that the distribution of $P(\tau)$ exhibits a bimodal pattern, with only a small proportion of τ values being close to zero. Furthermore, the Deoxys model tends to make confident predictions for the majority of samples, but occasionally its predictions remain hesitant. Based on the insights from Fig. 8, we have set the threshold parameters $\xi = 1$ and $\varepsilon = 15$, such that if $|\tau| < 1$ and its $\tau_u > 15$, we consider the model to be not confident, and therefore rollback to the legacy policy to implement a safer mitigation strategy. In production, approximately 2.3% of unhealthy requests trigger this fallback.

In a separate set of A/B testing, activating this fallback mechanism achieved similar VM downtime while reducing AIR by 18.1% compared to the group without it. This demonstrates its effectiveness in improving system reliability. It also validates that handcrafted policies are effective when the model is uncertain.

6.5 Overriding Redeploy with Reboot (RQ4)

We conducted an evaluation of the first action override mechanism in Sec. 4.3, where we choose to use Reboot instead of Redeploy, even when the estimator predicts that taking Redeploy would result in shorter VM downtime. The aim of this mechanism is to save more node resources that would be required by Redeploy, with minimal impact on downtime. In Fig. 9, we present the offline simulation results of the node and VM resource saving proportion, as well as their override ratio, within the requests that recommended the Redeploy action, with respect to the Redeploy override threshold ϖ. Recall that ϖ represents the maximum predicted downtime saving by taking Redeploy, when we override the Redeploy action in exchange for node resource. The figure indicates that by setting $\varpi = 1$, we can exchange 3% of the nodes requesting Redeploy while sacrificing only 1 unit of VM downtime. This results in substantial resource savings and enhances cluster capacity, presenting a favorable tradeoff between VM downtime and node resources. We have set this value in production for system capacity consideration.

[3]We hide the actual numbers of all metrics and only report the relative improvement, due to Microsoft's confidentiality requirement.

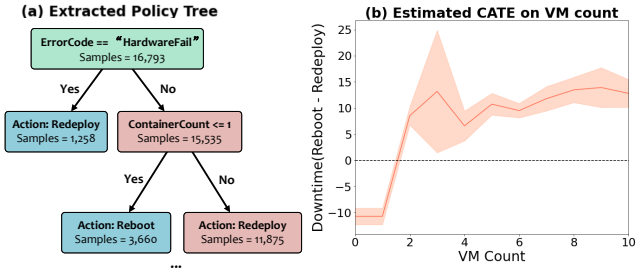

Figure 10: Policy interpreter. (a) The partial logical policy tree extracted; (b) The CATE conditioned on VM count.

6.6 Overriding Reboot with Redeploy (RQ5)

Additionally, we have implemented a feature called `RepeatCnt` to keep track of the number of unhealthy events that have occurred on the same node within the past 10 days. If the `RepeatCnt` exceeds a certain threshold, it indicates that the node has experienced repetitive failures that cannot be easily resolved by using Reboot alone. In such cases, we override the model's decision and opt for Redeploy regardless of its recommendation. Furthermore, we mark the node as unallocatable and initiate a human investigation to identify the root cause of the repetitive failures. In our production environment, we have empirically set the threshold for triggering this override mechanism at `RepeatCnt` > 10. As a result, approximately 8% of the unhealthy requests trigger this mechanism, out of which 13% were originally recommended with Reboot but were overridden in favor of Redeploy. Upon post-analysis of these cases, *it was observed that more than 96% of these nodes were susceptible to uncorrectable errors if the Reboot action was taken.* Our design effectively overrode this to Redeploy, successfully correcting the mitigation action. This overriding mechanism ensures that nodes experiencing recurrent failures receive the necessary mitigation measures, even when the model's prediction recommends a different course of action.

7 POLICY INTERPRETER

Finally, to enhance interpretability, the `EconML` library [5] provides a `SingleTreePolicyInterpreter` tool that offers a summary of the key diagnostic signals that explain the most significant differences in responsiveness to a mitigation action. This tool trains a shallow decision tree [55] using the treatment effect obtained from the `CauForeDML` on a small set of diagnostic signals. The decision tree learns an "if-else" like policy by splitting on cutoff points that maximize the treatment effect difference in each leaf. The resulting logical policy provides insights into why the `CauForeDML` makes a particular prediction, enhancing the interpretability and unveils the system behaviors in unhealthy events.

The initial two layers of the deduced logical policy are displayed in Fig 10 (a), with the subsequent layers omitted for brevity. Notably, the interpreter discloses that the foremost pivotal diagnostic signal driving the decision-making process for mitigation pertains to instances where the node reports a hardware failure error code. This rationale aligns with the inherent logic that, in the presence of hardware failures, resorting to a Reboot is unlikely to rectify the issue, thereby necessitating the adoption of the Redeploy action. In addition, the interpreter highlights the significance of the count of live VMs on the unhealthy node. When the VM count is less than or equal to 1, the recommended action is to take Reboot. On the other hand, when the VM count is greater than 1, the recommended action is to take Redeploy.

Based on the above findings, we further investigate the correlation of the treatment effect of the mitigations, conditioned on the alive VM count, as estimated by the Conditional Average Treatment Effect (CATE), as shown in Fig. 10 (b). The x-axis of the figure represents the VM count, and the y-axis represents the estimated downtime difference if taking Reboot instead of Redeploy. Interestingly, the CATE suggests that we should opt for Redeploy instead of Reboot with an increase in the VM count, as the CATE increases. It shows negative values at VM count= 1. This is aligned with the pivot point identified by the interpreter.

These findings prompt us to delve deeper into the their underlying factors. In general, a Reboot action tends to result in shorter downtime when the unhealthy signal is a false alarm, which occur frequently in cloud environments. However, there is a risk that the downtime will become significantly longer if the node indeed has unsolvable problems. On the other hand, the Redeploy action is considered safer as it relocates all VMs to a different node. Nonetheless, Redeploy increases the VM downtime when the node is actually healthy, as Reboot is faster in such cases. Conditioning the mitigation action on the number of VMs on the node allows for a tradeoff between risk and downtime. If there are many VMs on the node, it may be advisable to avoid the risk of Reboot, as the cost of a failed Reboot could be high. However, if there are only very few VMs, it may be beneficial to exploit the potential benefits of Reboot. These findings shed light on an interesting system tradeoff, and it is remarkable that this is learned by Deoxys.

8 LESSONS LEARNED AND LIMITATIONS

While Deoxys primarily focuses on determining "what to take" as a mitigation action when unhealthy events occur, it is important to note that some of these events may be false positives, such as when the loss of heartbeat signal is caused

by network issues rather than actual failures. In such cases, the node may not be truly unhealthy and may come back ready soon without any action needed. Mitigating these false positive events can lead to unnecessary VM interruptions and result in higher AIR. Therefore, developing a model to determine "when to take" action, *i.e.*, waiting for a certain duration before mitigation, can be a valuable addition to the framework. This approach can help improve both AVD and AIR, as unnecessary mitigation takes time and increases VM interruptions. We acknowledge that this is a potential area for future research.

We also recognize that, while historical unhealthy events, node status snapshots, and actions taken are crucial for understanding the system behavior. The current Deoxys does not consider this sequential information yet. This may limit the performance of the model. The next step is to build an online database to buffer such historical behaviors, allowing the model to leverage this information for more accurate action recommendations. This can potentially enhance the overall performance and effectiveness of the Deoxys.

9 RELATED WORK

Failure Mitigation plays a crucial role in modern cloud infrastructure [37], as it significantly impacts customers' experience [51, 74]. Several existing approaches have been proposed to address this challenge. Narya [34] is a proactive mitigation engine in Azure that utilizes multi-arm bandit techniques [60] to explore optimized mitigation policies in online environments. It has been further improved by NENYA [66], which integrates failure prediction [38, 40, 41] and mitigation into an end-to-end framework. Narya is considered as a non-contextual approach, which fails to model system heterogeneity given different node status. Moreover, these alternative methodologies necessitate online interactions with the live production environment for model updates, which in turn entail cold-start periods. Such operational requirements have the potential to induce performance regressions within the system. In contrast, Deoxys adopts an offline training approach, eschews frequent updates, and consistently yields dependable results. Consequently, this approach stands as a more favorable choice for production scenarios.

Causal Machine Learning Causal machine learning has emerged as a prominent topic in recent research due to its ability to automatically discover and estimate complex causal effects in high-dimensional data, even in the presence of unobserved confounders [3, 31, 72]. One of the key applications of causal machine learning is Individual Treatment Effect (ITE) estimation, which is also utilized in our proposed Deoxys [75]. ITE estimation has been widely employed in various domains such as recommender systems [11, 67] and healthcare applications [53, 56]. Commonly used methods for ITE estimation, such as S-learner [68], T-learner [49] and X-learner [76], are popular tools in the field. However, these methods often lack the ability to provide confidence intervals for their predictions, which could indicate the uncertainty of the models. In contrast, our proposed Deoxys approach employs DML [15] and causal forest [63] to deliver confidence scores for policy fallback. This improves the reliability of data-driven methods applied in cloud systems.

10 CONCLUSION

In this paper, we introduce Deoxys, a causal unhealthy node mitigation engine for large-scale cloud infrastructures. Deoxys leverages observational data to learn the causation between different mitigation actions and VM downtime, allowing it to choose the most effective action to minimize downtime during unhealthy events. The reliability and trustworthiness of Deoxys are ensured through the incorporation of policy fallback and action override mechanisms, improving the robustness of the system. Furthermore, it has the capability to extract logical rules from data, elucidating its decision-making process and enhancing system interpretability. Notably, we deployed Deoxys as a key unhealthy node mitigation engine in a large-scale cloud infrastructure at Microsoft. Our observations have shown that, on average, Deoxys reduces VM downtime by 53% compared to a legacy mitigation policy, while leading 49.5% lower VM interruption rate. This demonstrates the effectiveness of Deoxys in enhancing the performance and stability of the cloud computing platform, thereby providing better customer experience and potentially leading to huge business value.

REFERENCES

[1] [n. d.]. Cloud Outages: Causes, Consequences, Prevention, Recovery. https://www.informationweek.com/cloud/cloud-outages-causes-consequences-prevention-recovery. Accessed: 2023-04-26.

[2] Anup Agarwal, Shadi Noghabi, Íñigo Goiri, Srinivasan Seshan, and Anirudh Badam. 2023. Unlocking unallocated cloud capacity for long, uninterruptible workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 457–478.

[3] Susan Athey and Guido W Imbens. 2015. Machine learning methods for estimating heterogeneous causal effects. *stat* 1050, 5 (2015), 1–26.

[4] Susan Athey, Julie Tibshirani, and Stefan Wager. 2019. Generalized random forests. (2019).

[5] Keith Battocchi, Eleanor Dillon, Maggie Hei, Greg Lewis, Paul Oka, Miruna Oprescu, and Vasilis Syrgkanis. 2019. EconML: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. https://github.com/py-why/EconML. Version 0.x.

[6] Alan W Beggs. 2005. On the convergence of reinforcement learning. *Journal of economic theory* 122, 1 (2005), 1–36.

[7] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[8] Peter Bühlmann and Sara Van De Geer. 2011. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media.

[9] George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. 2004. Microreboot–a technique for cheap recovery.

arXiv preprint cs/0406005 (2004).

[10] Chandranil Chakraborttii and Heiner Litz. 2020. Improving the accuracy, adaptability, and interpretability of SSD failure prediction models. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 120–133.

[11] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2023. Bias and debias in recommender system: A survey and future directions. *ACM Transactions on Information Systems* 41, 3 (2023), 1–39.

[12] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* 1, 4 (2015), 1–4.

[13] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2023. Empowering Practical Root Cause Analysis by Large Language Models for Cloud Incidents. *arXiv preprint arXiv:2305.15778* (2023).

[14] Yuhang Chen, Chaoyun Zhang, Minghua Ma, Yudong Liu, Ruomeng Ding, Bowen Li, Shilin He, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. ImDiffusion: Imputed Diffusion Models for Multivariate Time Series Anomaly Detection. *arXiv preprint arXiv:2307.00754* (2023).

[15] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. 2018. Double/debiased machine learning for treatment and structural parameters.

[16] Carlos Colman-Meixner, Chris Develder, Massimo Tornatore, and Biswanath Mukherjee. 2016. A survey on resiliency techniques in cloud computing infrastructures and applications. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2244–2281.

[17] Thomas J DiCiccio and Bradley Efron. 1996. Bootstrap confidence intervals. *Statistical science* 11, 3 (1996), 189–228.

[18] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1762–1773.

[19] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Everything of thoughts: Defying the law of penrose triangle for thought generation. *arXiv preprint arXiv:2311.04254* (2023).

[20] Ifeanyi P Egwutuoha, Shiping Chen, David Levy, Bran Selic, and Rafael Calvo. 2012. A proactive fault tolerance approach to High Performance Computing (HPC) in the cloud. In *2012 Second International Conference on Cloud and Green Computing*. IEEE, 268–273.

[21] Patricia Takako Endo, Guto Leoni Santos, Daniel Rosendo, Demis Moacir Gomes, André Moreira, Judith Kelner, Djamel Sadok, Glauco Estácio Gonçalves, and Mozhgan Mahloo. 2017. Minimizing and managing cloud failures. *Computer* 50, 11 (2017), 86–90.

[22] J-Cl Falmagne and J-P Doignon. 1988. A Markovian procedure for assessing the state of a system. *Journal of Mathematical Psychology* 32, 3 (1988), 232–258.

[23] Peter Garraghan, Paul Townend, and Jie Xu. 2014. An empirical failure-analysis of a large-scale cloud computing environment. In *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE, 113–120.

[24] Sukhpal Singh Gill, Inderveer Chana, Maninder Singh, and Rajkumar Buyya. 2019. RADAR: Self-configuring and self-healing in resource management for enhancing quality of cloud services. *Concurrency and Computation: Practice and Experience* 31, 1 (2019), e4834.

[25] TianZhang He and Rajkumar Buyya. 2023. A taxonomy of live migration management in cloud computing. *Comput. Surveys* 56, 3 (2023), 1–33.

[26] Xiangnan He, Yang Zhang, Fuli Feng, Chonggang Song, Lingling Yi, Guohui Ling, and Yongdong Zhang. 2023. Addressing confounding feature issue for causal recommendation. *ACM Transactions on Information Systems* 41, 3 (2023), 1–23.

[27] Peng Huang, Chuanxiong Guo, Jacob R Lorch, Lidong Zhou, and Yingnong Dang. 2018. Capturing and enhancing in situ system observability for failure detection. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 1–16.

[28] Yuxuan Jiang, Chaoyun Zhang, Shilin He, Zhihao Yang, Minghua Ma, Si Qin, Yu Kang, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, et al. 2023. Xpert: Empowering incident management with query recommendations via large language models. *arXiv preprint arXiv:2312.11988* (2023).

[29] Marshall M Joffe and Paul R Rosenbaum. 1999. Invited commentary: propensity scores. *American journal of epidemiology* 150, 4 (1999), 327–333.

[30] Yonghan Jung, Jin Tian, and Elias Bareinboim. 2021. Estimating identifiable causal effects through double machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 12113–12122.

[31] Jean Kaddour, Aengus Lynch, Qi Liu, Matt J Kusner, and Ricardo Silva. 2022. Causal machine learning: A survey and open problems. *arXiv preprint arXiv:2206.15475* (2022).

[32] Edward H Kennedy. 2022. Semiparametric doubly robust targeted double machine learning: a review. *arXiv preprint arXiv:2203.06469* (2022).

[33] Sören R Künzel, Jasjeet S Sekhon, Peter J Bickel, and Bin Yu. 2019. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the national academy of sciences* 116, 10 (2019), 4156–4165.

[34] Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, et al. 2020. Predictive and adaptive failure mitigation to avert production cloud VM interruptions. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 1155–1170.

[35] Greg Lewis and Vasilis Syrgkanis. 2020. Double/debiased machine learning for dynamic treatment effects via g-estimation. *arXiv preprint arXiv:2002.07285* (2020).

[36] Greg Lewis and Vasilis Syrgkanis. 2021. Double/debiased machine learning for dynamic treatment effects. *Advances in Neural Information Processing Systems* 34 (2021), 22695–22707.

[37] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, et al. 2020. Gandalf: An Intelligent, End-To-End Analytics Service for Safe Deployment in Large-Scale Cloud Infrastructure.. In *NSDI*, Vol. 20. 389–402.

[38] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 480–490.

[39] Jun Liu, Chaoyun Zhang, Jiaxu Qian, Minghua Ma, Si Qin, Chetan Bansal, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2024. Large Language Models can Deliver Accurate and Interpretable Time Series Anomaly Detection. *arXiv preprint arXiv:2405.15370* (2024).

[40] Yudong Liu, Hailan Yang, Pu Zhao, Minghua Ma, Chengwu Wen, Hongyu Zhang, Chuan Luo, Qingwei Lin, Chang Yi, Jiaojian Wang, et al. 2022. Multi-task Hierarchical Classification for Disk Failure

Prediction in Online Service Systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3438–3446.

[41] Minghua Ma, Yudong Liu, Yuang Tong, Haozhe Li, Pu Zhao, Yong Xu, Hongyu Zhang, Shilin He, Lu Wang, Yingnong Dang, et al. 2022. An empirical investigation of missing data handling in cloud node failure prediction. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1453–1464.

[42] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.

[43] Marloes H Maathuis and Diego Colombo. 2015. A generalized backdoor criterion. (2015).

[44] Daniele Miorandi, Alessandra Rizzardi, Sabrina Sicari, and Alberto Coen-Porisini. 2019. Sticky policies: A survey. *IEEE Transactions on Knowledge and Data Engineering* 32, 12 (2019), 2481–2499.

[45] Stephen L Morgan and Christopher Winship. 2015. *Counterfactuals and causal inference*. Cambridge University Press.

[46] Mina Nabi, Maria Toeroe, and Ferhat Khendek. 2016. Availability in the cloud: State of the art. *Journal of Network and Computer Applications* 60 (2016), 54–67.

[47] Jerzy Neyman. 1959. Optimal asymptotic tests of composite hypotheses. *Probability and statsitics* (1959), 213–234.

[48] Xinkun Nie and Stefan Wager. 2021. Quasi-oracle estimation of heterogeneous treatment effects. *Biometrika* 108, 2 (2021), 299–319.

[49] Diego Olaya, Jonathan Vásquez, Sebastián Maldonado, Jaime Miranda, and Wouter Verbeke. 2020. Uplift Modeling for preventing student dropout in higher education. *Decision support systems* 134 (2020), 113320.

[50] Miruna Oprescu, Vasilis Syrgkanis, and Zhiwei Steven Wu. 2019. Orthogonal random forest for causal inference. In *International Conference on Machine Learning*. PMLR, 4932–4941.

[51] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kiciman, Matthew Merzbacher, et al. 2002. *Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies*. Technical Report. Citeseer.

[52] Judea Pearl. 2009. *Causality*. Cambridge university press.

[53] Mattia Prosperi, Yi Guo, Matt Sperrin, James S Koopman, Jae S Min, Xing He, Shannan Rich, Mo Wang, Iain E Buchan, and Jiang Bian. 2020. Causal inference and counterfactual prediction in machine learning for actionable healthcare. *Nature Machine Intelligence* 2, 7 (2020), 369–375.

[54] Donald B Rubin and Neal Thomas. 1996. Matching using estimated propensity scores: relating theory to practice. *Biometrics* (1996), 249–264.

[55] S Rasoul Safavian and David Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics* 21, 3 (1991), 660–674.

[56] Pedro Sanchez, Jeremy P Voisey, Tian Xia, Hannah I Watson, Alison Q O'Neil, and Sotirios A Tsaftaris. 2022. Causal machine learning for healthcare and precision medicine. *Royal Society Open Science* 9, 8 (2022), 220638.

[57] Uri Shalit, Fredrik D Johansson, and David Sontag. 2017. Estimating individual treatment effect: generalization bounds and algorithms. In *International conference on machine learning*. PMLR, 3076–3085.

[58] Amit Sharma and Emre Kiciman. 2020. DoWhy: An End-to-End Library for Causal Inference. *arXiv preprint arXiv:2011.04216* (2020).

[59] Rachee Singh, Muqeet Mukhtar, Ashay Krishna, Aniruddha Parkhi, Jitendra Padhye, and David Maltz. 2021. Surviving switch failures in cloud datacenters. *ACM SIGCOMM Computer Communication Review* 51, 2 (2021), 2–9.

[60] Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning* 12, 1-2 (2019), 1–286.

[61] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.

[62] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. 2010. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*. 193–204.

[63] Stefan Wager and Susan Athey. 2018. Estimation and inference of heterogeneous treatment effects using random forests. *J. Amer. Statist. Assoc.* 113, 523 (2018), 1228–1242.

[64] Clifford H Wagner. 1982. Simpson's paradox in real life. *The American Statistician* 36, 1 (1982), 46–48.

[65] Lu Wang, Chaoyun Zhang, Ruomeng Ding, Yong Xu, Qihang Chen, Wentao Zou, Qingjun Chen, Meng Zhang, Xuedong Gao, Hao Fan, et al. 2023. Root Cause Analysis for Microservice Systems via Hierarchical Reinforcement Learning from Human Feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5116–5125.

[66] Lu Wang, Pu Zhao, Chao Du, Chuan Luo, Mengna Su, Fangkai Yang, Yudong Liu, Qingwei Lin, Min Wang, Yingnong Dang, et al. 2022. NENYA: Cascade Reinforcement Learning for Cost-Aware Failure Mitigation at Microsoft 365. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4032–4040.

[67] Wenjie Wang, Fuli Feng, Xiangnan He, Xiang Wang, and Tat-Seng Chua. 2021. Deconfounded recommendation for alleviating bias amplification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1717–1725.

[68] Guoqiang Xu, Cunxiang Yin, Yuchen Zhang, Yuncong Li, Yancheng He, Jing Cai, and Zhongyu Wei. 2022. Learning discriminative representation base on attention for uplift. In *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, China, May 16–19, 2022, Proceedings, Part III*. Springer, 200–211.

[69] Chaoyun Zhang, Marco Fiore, Iain Murray, and Paul Patras. 2021. CloudLSTM: A recurrent neural model for spatiotemporal point-cloud stream forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10851–10858.

[70] Chaoyun Zhang, Marco Fiore, Cezary Ziemlicki, and Paul Patras. 2020. Microscope: mobile service traffic decomposition for network slicing as a service. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[71] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. UFO: A UI-Focused Agent for Windows OS Interaction. *arXiv preprint arXiv:2402.07939* (2024).

[72] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials* 21, 3 (2019), 2224–2287.

[73] Chaoyun Zhang, Kai Wang, Hao Chen, Ge Fan, Yingjie Li, Lifang Wu, and Bingchao Zheng. 2022. Quickskill: Novice skill estimation in online multiplayer games. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3644–3653.

[74] Qiao Zhang, Guo Yu, Chuanxiong Guo, Yingnong Dang, Nick Swanson, Xinsheng Yang, Randolph Yao, Murali Chintalapati, Arvind Krishnamurthy, and Thomas Anderson. 2018. Deepview: Virtual disk failure diagnosis and pattern detection for azure. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 519–532.

[75] Weijia Zhang, Jiuyong Li, and Lin Liu. 2021. A unified survey of treatment effect heterogeneity modelling and uplift modelling. *ACM*

Computing Surveys (CSUR) 54, 8 (2021), 1–36.

[76] Zhenyu Zhao and Totte Harinen. 2019. Uplift modeling for multiple treatments with cost optimization. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 422–431.