ChatGPT

Root Cause Analysis for Data Center Temperature Anomalies

Overview and Requirements

Monitoring a data center's ambient temperature and identifying **root causes of anomalies** involves two core steps: (1) detecting when the temperature deviates abnormally (anomaly detection), and (2) analyzing **why** that deviation occurred (root cause analysis, or RCA). In our case, the solution must run **locally on a MacBook** (no cloud dependency), integrate with a Python/Streamlit app, and emphasize **transparency and interpretability**. Key capabilities include:

- **Custom Time Range & Thresholds:** The user specifies a time window and acceptable temperature limits. The system flags all ambient temperature readings outside the tolerable range (or otherwise abnormal patterns in that period).
- **Anomaly Detection:** Reliable methods to catch temperature spikes/dips – possibly combining simple threshold rules with learned models for more subtle anomalies.
- **Root Cause Identification:** For each detected anomaly, the system pinpoints likely causes (e.g. *AC unit off*, *door opened*, *external temperature spike*, *IT load surge*). It should rank causes by likelihood and provide a confidence score and explanation for each.
- **Local & Python-Friendly:** All components should run on a MacBook (no specialized hardware needed) and be usable within Python (via libraries or local services).
- **Interpretability:** Solutions should offer insight into **why** an anomaly is flagged and why a cause is suspected – supporting trust and easy debugging.

Below, we discuss best practices for anomaly detection and RCA in time-series data, and review leading tools and frameworks that meet these criteria. We consider rule-based approaches, machine learning libraries, causal analysis frameworks, and integrated platforms, highlighting their features, ease of implementation, and suitability for a local Python environment.

Best Practices in Anomaly Detection & Root Cause Analysis

In time-series monitoring (like data center temperatures), anomaly detection and RCA work hand-in-hand. **Anomaly detection** is the "watchdog" that alerts us something is wrong, while **root cause analysis** is the "detective" that investigates why it happened [1]. Effective RCA typically involves:

- **Multi-metric Correlation:** When an ambient temperature anomaly occurs, examine other metrics/sensors (AC unit status, door sensors, external weather, IT load, etc.) for concurrent anomalies. Often the root cause will show up as a deviation in a related metric around the same time [2]. For example, a spike in ambient temperature might coincide with an AC unit turning off or an exterior temperature peak.
- **Domain Knowledge & Rules:** Leverage known relationships (causal links) between variables. For instance, *"If AC is off for >5 minutes, ambient temperature will rise"* can be a rule. Simple **rule-based checks** (like threshold alarms for AC status or door open events) are **interpretable** and fast. However, purely manual rules can be brittle and hard to scale [3]. A hybrid approach is recommended: use domain-based rules to catch obvious causes and lightweight ML models to catch complex or unforeseen patterns.

- **Statistical Anomaly Detection:** Establish what "normal" temperature behavior is (accounting for daily cycles or seasonal trends) and flag statistically significant deviations. A common rule of thumb is using standard deviation bands or percentile thresholds. This is simple to implement but must be tuned carefully for environments with evolving baselines [4] [5].
- **Machine Learning Detection:** Unsupervised ML models like Isolation Forest, Local Outlier Factor, or One-Class SVM can automatically learn the normal pattern of temperature and detect oddities [6]. ML can handle subtle anomalies and multivariate patterns that rules might miss. Crucially, ML can also **correlate multiple data streams**, which aids root cause analysis by linking anomalies across many sensors [3]. As Red Hat's observability team notes, ML models can "scale across large sets of metrics and… enable correlation of vast amounts of data, providing superior root cause analysis and earlier detection of anomalies—often before service degradation thresholds are reached" [3].
- **Causal Analysis:** Go beyond correlation by using causal inference or graphical models. If you can encode the cause-effect structure of your system (e.g. external temperature → ambient temperature, AC power → ambient temperature, door position → ambient temperature), you can perform targeted tests to see which cause best explains the anomaly. This often yields a **confidence score** per cause. Modern RCA frameworks use **causal graphs** and algorithms to localize root causes by traversing cause-effect chains [2]. This approach is powerful for complex systems where multiple factors interplay.

In practice, a **hybrid strategy** is often best [7]: for example, use simple threshold rules to catch any temperature reading outside safe bounds (a direct anomaly), combine that with an ML anomaly detector for pattern-based anomalies (like an unusual rate of temperature rise), then apply a root cause algorithm that checks each potential factor (AC, door, external weather, etc.) for concurrent anomalies or causal influence. The following figure illustrates how anomaly detection integrates with a causal RCA pipeline:

Example of an anomaly detection and RCA pipeline. The monitoring system detects an anomalous metric (e.g., ambient temperature spike) and triggers an RCA module. The RCA module uses multiple data sources and a causal graph (built from expert knowledge and data) to compute root cause scores and identify the top-K likely cause metrics [2].

In the sections below, we explore specific tools and libraries that implement these principles, focusing on those that are easy to run locally and integrate with Python.

Rule-Based and Statistical Methods

Threshold Rules & Basic Stats: The simplest approach is to use user-defined thresholds on the ambient temperature. If temperature exceeds (or drops below) the specified safe range, flag an anomaly. This can be implemented with a few lines of Python (e.g., using pandas to filter out-of-bound values). While straightforward and **fully transparent** (easy to explain: "Temperature exceeded X°C"), this method won't catch more subtle anomalies (like a sudden jump that is still within limits, or anomalous fluctuations). It's best used as an initial filter.

ADTK (Anomaly Detection Toolkit): An open-source Python toolkit specifically for **unsupervised and rule-based time series anomaly detection** [8]. ADTK makes it easy to define rules like level shifts, persistently high values, seasonal pattern deviations, etc. For example, one can set up: a *ThresholdDetector* for absolute limits, a *VolatilityShiftDetector* for sudden variance changes, or a *PersistenceDetector* for readings that stay high for too long. ADTK is lightweight, works on pure Python, and is highly interpretable (each detector is essentially a rule or simple model) [8]. In our scenario, ADTK could be configured to combine a threshold rule on ambient temperature with a rule that

monitors the difference between ambient and external temperature (to catch cases where outside heat might be influencing indoor temperature). This rule-based approach is **easy to implement on a MacBook** and doesn't require any cloud services. Its limitation is that you must manually choose and tune the rules, but it provides a solid, transparent baseline.

Statistical Anomaly Detection: Methods like moving average and standard deviation can be implemented with libraries like SciPy or pandas. For instance, calculate a rolling mean and standard deviation of ambient temp over a period and flag points outside 3σ as anomalies (a common practice for outlier detection [9]). These are essentially simple cases of ADTK's capabilities. They are fast and local, but as mentioned, defining "normal" via static statistics can be tricky if the data has trends or periodic cycles [10]. If the data center has daily cooling cycles, a static threshold might over-alert; one might need to separate day vs night baseline or use a more adaptive method (like an exponentially weighted moving average control chart).

When to use: Rule-based and statistical methods are ideal when you have strong domain knowledge of acceptable ranges and want maximum interpretability. They run easily within a Streamlit app (just Python code, possibly using ADTK or custom logic). We recommend using them as a first layer (to catch gross anomalies and feed into the cause analysis). However, for **more accuracy and fewer false alarms**, it's beneficial to augment them with ML-based methods as described next.

Machine Learning-Based Anomaly Detection

PyCaret (Anomaly Detection Module): PyCaret is a user-friendly AutoML library, and it includes an unsupervised anomaly detection module for outlier detection on tabular data [11] [12]. PyCaret essentially provides a high-level API to algorithms from the popular PyOD library (Python Outlier Detection) [13]. For example, with a few lines of code you can train an Isolation Forest or KNN detector on the ambient temperature time-series. It will assign an "anomaly score" to each timestamp and label the most extreme points as outliers. The benefit of PyCaret is **ease of use** – you can compare multiple detectors (Isolation Forest, LOF, One-class SVM, etc.) with a single `setup()` call and `compare_models()`. It's fully Python and runs on a Mac (no special hardware needed, as these models are relatively light). PyCaret also can output an **anomaly score** per point, which you might translate into a confidence level for how abnormal a reading is [13].

Implementation: In a Streamlit app, you could let PyCaret train on the selected time range's data (or a baseline period) to learn normal behavior. Then apply the model to the same range to identify anomalies. For each anomaly, since PyCaret is a general ML tool, it doesn't automatically tell you *why* that point is anomalous – it just flags it. You would then need to investigate other metrics at that timestamp to infer root causes. However, one strategy is to use PyCaret's integration with SHAP (SHapley values) on anomaly detection models (if supported) or simply examine feature values: e.g., if you include multiple features (ambient temp, external temp, AC status) in the anomaly model, an Isolation Forest might implicitly use all features to detect anomalies. You could then inspect which feature was most "extreme" for that anomalous point as a clue to the cause. This is not as straightforward as dedicated RCA tools, but it's a possible approach.

PyOD: This is the library under the hood of PyCaret's anomaly module [13]. PyOD can be used directly for more control. It offers 20+ outlier detection algorithms (many the same as PyCaret) and is well-maintained. If you prefer not to use the PyCaret wrapper, you can import PyOD (e.g., `from pyod.models.iforest import IForest`) and use it directly in Python. PyOD is also easy to integrate and works locally. The difference is PyOD will require you to handle data preprocessing and model selection manually, whereas PyCaret automates some of that. Both PyCaret and PyOD are

reliable; since they rely on mature algorithms (Isolation Forest, etc.), they are quite **accurate for capturing anomalies** when tuned properly.

Time-Series Specific ML: Some libraries like **Merlion** (by Salesforce) and **Kats** (by Facebook) provide anomaly detection tailored to time-series data. For example, Merlion is an open-source framework with a suite of anomaly detection models (from statistical to deep learning) unified under one API [14] [15]. Merlion even includes threshold-based detectors and can ensemble multiple models. These can run locally (Merlion is pure Python, though some models may use PyTorch for deep learning). If your ambient temperature data has seasonality or trend, Merlion's forecasting-based anomaly detection (which uses a model like Prophet to predict expected values and flags deviations) could be useful. Similarly, Kats provides detectors like Prophet or statistical control charts as well. Using these libraries might be overkill for a single metric, but if you plan to extend to more signals (humidity, power usage, etc.), they offer a scalable way to manage many anomaly detectors.

Recommendation: For **ease-of-implementation** in a Streamlit app, PyCaret is a friendly choice to quickly get anomaly flags, but keep in mind you'll need to supplement it with a method to interpret root causes. PyCaret/PyOD can reliably catch outliers, and you can then cross-reference those times with other sensor readings. Another lightweight route is using **ADTK's ML components** – ADTK isn't just rules; it also has some outlier models and can incorporate things like ARIMA-based detectors. That said, if you anticipate complex anomalies, investing time in a library like Merlion (which even has an experimental integration with PyRCA for RCA tasks [16]) might pay off. In summary, ML detectors increase detection accuracy and can catch anomalies that simple thresholds miss, forming a strong basis for subsequent RCA.

Causal and Graph-Based RCA Solutions

Once anomalies are detected, pinpointing the root cause is the challenging part. This is where specialized RCA libraries shine. These tools use combinations of statistical tests, causal graphs, and search algorithms to automate the detective work that an engineer would normally do manually (checking each related metric). Below are some of the most relevant solutions that run locally in Python:

PyRCA (Python Root Cause Analysis by Salesforce)

PyRCA is an open-source library designed specifically for RCA in IT systems [17]. It provides an end-to-end framework: you feed it time-series data (metrics) and it can *discover causal relationships* and *identify root causes* when incidents occur [18] [19]. PyRCA supports multiple state-of-the-art RCA algorithms, accessible via a unified API [18] [20]. Notably, it includes:

- *Graph-based methods:* You can construct a **causal graph** of your metrics (either manually, from domain knowledge, or using PyRCA's graph discovery tools) and then use algorithms like Bayesian Network inference or Random Walk to find which node (metric) in the graph most likely caused the anomaly [21] [22]. For example, in our case, nodes might be Ambient Temp, External Temp, AC Status, Door Status, IT Load. If ambient temp is anomalous, PyRCA can analyze the graph and metric histories to score each node by how likely it is the source of the anomaly.
- *Anomalous metric detection:* PyRCA can also simply look for other metrics that showed anomalies in the same time window (this is akin to the ε-diagnosis method) [21]. If, say, the door sensor reading was also abnormal when the temperature spiked, that metric gets flagged as a possible cause.

- *Hybrid approach:* You can combine the above – e.g., first filter metrics by whether they had anomalies, then use the causal graph to refine which of those anomalies could actually lead to the ambient temp issue.

PyRCA is **built in Python** and easy to install (`pip install sfr-pyrca`). It's compatible with pandas DataFrames and integrates well into a Python app. Importantly, it emphasizes **transparency**: you can incorporate **expert knowledge** in the form of graph structure or constraints to guide the RCA [20] [23]. This means the model's reasoning is not a black box; it's rooted in the known system architecture or physics. PyRCA also provides an interactive dashboard for visualizing data and causal graphs [24] [25], but you can use its core logic headless within Streamlit.

For output, PyRCA will give you a ranked list of likely root cause metrics with associated scores (which you can interpret as confidence levels). For instance, it might output: *"AC_Power_Failure: 95% likelihood; Door_Open_Event: 40% likelihood"*, with an explanation like *"AC_Power_Failure showed an anomalous drop 10 minutes before ambient temperature exceeded threshold"*. Under the hood, these scores come from the chosen RCA algorithm (Bayesian inference might output probabilities, hypothesis testing might output significance levels, etc.). PyRCA's emphasis on **multiple algorithms** means you can compare approaches for reliability [22]. Overall, PyRCA is a **strong candidate** for our needs: it's local, Python-based, designed for exactly this kind of problem, and supports interpretability (via causal graphs and knowledge integration).

ProRCA (Causal Pathway Root Cause Analysis)

ProRCA (available as the `profitops-rca` package) is a very recent open-source framework (initial release in 2025) that focuses on *causal pathway discovery* for anomalies [26]. It was developed to go beyond simple correlation or feature importance methods by leveraging **structural causal modeling** to trace multi-hop causal chains from root cause to observed anomaly [26]. In essence, ProRCA builds a causal model of the system and uses it to explain anomalies. Key features:

- It integrates an anomaly detection step (currently using the ADTK library's IQR rule by default) to first spot anomalies in a time-series [27]. In our scenario, that would flag the ambient temperature spike.
- It then requires a **causal graph** (which you can define, or potentially learn) describing relationships between variables. With our domain, we'd define a graph connecting AC, door, outside temp, IT load, and ambient temp.
- Using its `CausalRootCauseAnalyzer`, ProRCA will **discover and rank causal pathways** that could lead to the anomaly [28]. This means it doesn't just point to one metric – it can identify a chain like *External Heat Wave → AC Overload → AC Off → Ambient Temp Rise* if the data supports that sequence. It combines *structural anomaly scoring* (checking which part of the causal model had abnormal behavior) with *noise-based attribution* (seeing how "noise" in each cause would affect the target) to rank these pathways [29] [30].
- It outputs results that include the paths and some form of importance score. These scores can be interpreted as confidence – higher means that pathway strongly explains the anomaly. ProRCA also includes a visualization module to draw the causal graph highlighting the most significant paths (with gradient colors indicating importance) [28].

ProRCA is Python-based and built on top of the PyWhy/DoWhy causal inference libraries [31], meaning it's leveraging robust academic algorithms (the authors cite an upcoming paper for the methodology). Because it's new, **ease of use** might not be as polished as PyRCA, but the documentation provides examples. Installing it (`pip install profitops-rca`) brings in everything needed (it's lightweight, only ~20KB of code plus dependencies like networkx, ADTK, etc.) [32]. Running ProRCA in a Streamlit app

is feasible: you'd load your data, define the causal structure (could even be hard-coded for known relations), detect anomalies (it has a helper for this), then call the analyzer. The **benefit of ProRCA** is its explicit causal chain output, which provides a very clear explanation ("A led to B led to C"). This suits complex incidents where multiple factors conspire. In our context, if, say, both a door opened and an AC issue occurred, ProRCA might identify that dual chain.

In summary, ProRCA is a cutting-edge solution aligning perfectly with **local, interpretable RCA**. It might require a bit more effort to set up the causal model, but it yields rich insights. Given its recency, one should validate its stability, but it is certainly **one of the most accurate and explanatory approaches** given it truly attempts to capture causation rather than just correlation [26].

DoWhy (PyWhy Causal Toolkit)

DoWhy is a general Python library for causal inference (not specific to anomalies, but applicable). It was extended by AWS researchers with new algorithms to support automated root cause analysis of observed changes [33]. In a January 2023 AWS Open Source Blog, they describe how DoWhy can identify root causes of an unexpected metric change by using a **causal graph and causal impact analysis** [34] [35]. Essentially, you provide DoWhy with a structural causal model (SCM) – a directed graph plus some statistical models of how each cause affects effects. Then, given an observed anomaly (e.g., profit dropped or temperature spiked), DoWhy's RCA features can evaluate which input factor's change would most likely explain that outcome [36] [37]. This often involves computing something like a *causal contribution score* for each parent node of the anomaly.

Applying DoWhy directly would mean: model the data center system in a causal DAG (using e.g. `networkx` as shown in the AWS blog example [38] [39]) – which is similar to what we'd do for PyRCA/ProRCA. Then use DoWhy's functions to simulate or infer what must have changed to cause the ambient temperature to go out of bounds. For example, you might use DoWhy to estimate *"if AC had been on vs off, what difference in ambient temp"* or use their provided algorithms that automatically scan through possible variables to find the culprit. This is a powerful approach grounded in causal theory, and it can give a **quantitative confidence (like probability or effect size)** for each cause. DoWhy is purely Python and will run on a Mac (it may use some statistical computing libraries but nothing cloud-based). However, it is a lower-level toolkit compared to PyRCA/ProRCA: you'll be writing more code and making more decisions (like how to model each relation).

Recommendation: If you prefer a more manual but flexible causal analysis and have expertise in causal modeling, DoWhy (or its newer PyWhy components) is a great resource. AWS's contribution shows that with a "few lines of code" it can automatically find root causes in complex systems [40]. But for most users, a higher-level library like PyRCA or ProRCA will be easier to implement, since they wrap around such causal analysis with more automation. Notably, ProRCA actually builds on DoWhy internally [31], so using ProRCA might give you the benefit of DoWhy's algorithms with simpler API.

Other Noteworthy Solutions

- **ELK Stack (Elastic Stack) with AIOps/ML:** The Elastic Stack (Elasticsearch + Kibana) is traditionally a logging and metrics platform, but it includes machine learning features for anomaly detection and correlations. Elastic can model time-series metrics to automatically detect anomalies (by learning normal patterns of the data) [41] and also perform **log outlier detection**. In an observability context, Elastic's AIOps features will highlight an anomaly in a metric and then help **pinpoint potential root causes** by finding related anomalies in other metrics or unusual log events at that time [42]. For example, if ambient temperature spiked, an Elastic ML job could catch that, and Kibana might surface that *"at the same time, log events*

indicate CRAC Unit 2 reported an error" or *"power usage anomaly also detected in Rack 3"*. Elastic achieves this through its anomaly detection jobs and a feature called **APM correlations** or **log categorization**, which cluster log messages to find common factors during incidents [43] [44].

To use Elastic locally, you'd have to run an Elasticsearch instance (which can be done on a Mac, though it's heavier than a simple Python script). You could feed your sensor data and event logs into it. Kibana's interface would then allow you to set up anomaly detection on the ambient temperature and related metrics. The results can be accessed via the Kibana UI or via Elasticsearch APIs. Integration into a Streamlit app is not direct – you might query Elasticsearch from Python to get anomaly results or cause indicators. Elastic's solution is very **robust and "production-grade"**, benefiting from built-in ML and the ability to combine logs, metrics, traces in one place. It's reliable and fairly accurate (since it uses advanced probabilistic models under the hood). It's also interpretable in the sense that it will tell you *which fields* in logs or *which related metric* had an anomaly that correlates with the primary issue. However, it's not a simple pip install; it requires running services. If you already have an ELK stack or don't mind the overhead, this is a powerful option. Otherwise, for a lightweight local solution, pure Python libraries might be easier.

- **Chaos Genius:** This was an open-source "business observability" tool that offered automated outlier detection and root cause analysis for metrics. It could be used to monitor metrics and then perform "deep drills" to find which dimensions or factors changed [45] [46]. For example, if a metric (like power usage cost) went up, it could break down which region or product contributed. In our context, we don't have high-dimensional data, but one could imagine using it to break down anomalies by, say, location or device if multiple sensors were involved. Chaos Genius included **anomaly detection models** (Prophet, etc.) and had an *"Automated Root Cause Analysis"* feature on their roadmap [47]. It ran as a local web service (via Docker) with a UI and could integrate with data sources. **Note:** Chaos Genius's repository is now archived (no longer actively maintained) [48]. While it's an example of an open-source RCA tool, using an archived project is not ideal for critical tasks. Still, it demonstrates the concept of an end-to-end local solution. The modern alternatives (PyRCA, ProRCA) are more focused and code-centric rather than a full web platform.

- **Hybrid Custom Solution:** It's worth mentioning that one can build a custom RCA solution by combining simpler components. For instance: use **PyOD/PyCaret** to detect an anomaly, then write a small routine to check a list of potential causes. The routine could use correlation (e.g., see if external temperature was above its typical range during the incident) or simple logic (if door sensor indicates "open" at anomaly time, flag that). You could even train a **classification model** offline: label past anomalies with their known causes and train a decision tree to predict cause from sensor readings – the tree's logic would be interpretable (this is feasible if you have historical incident data). Such a DIY approach might leverage scikit-learn (for classifiers or regression to model relationships) and produce a decision rule like "IF AC=Off AND External_Temp>35°C THEN AmbientTempSpike (confidence 90%)". While this requires more manual effort and is case-specific, it may be perfectly sufficient for a controlled environment with a handful of known factors. The advantage is you have full control and can ensure it runs locally (it's just Python code). The disadvantage is reinventing some wheels that the above libraries already provide (and tested more thoroughly).

Comparison of RCA Solutions and Tools

To summarize the options and how they meet our criteria, the table below compares key solutions:

Solution / Approach	Methodology	Features	Interpretability	Local Deployment	Ease of Integration
Manual Rules + ADTK	Rule-based thresholds & detectors	Simple thresholds, pattern rules (e.g. persistence, level shift via ADTK) [8] . No ML needed.	Very High – rules are transparent (e.g., “temp > X°C for 5 min”).	100% local (pure Python).	Easy (few lines per rule; ADTK is plug-and-play).
PyCaret (PyOD)	Unsupervised ML (PyOD models)	Automated outlier detection (Isolation Forest, LOF, etc.) with PyCaret’s simple API [11] . Can detect subtle anomalies.	Moderate – Flags anomalies but doesn’t explain cause by itself. (Underlying algorithms are known, but need extra analysis for cause.)	Local (Python); no cloud required.	Very easy for anomaly detection (AutoML style). Needs custom coding to map anomalies to causes.
PyRCA	Metric correlation + Causal graph RCA	End-to-end RCA: anomaly detection (basic stats) + causal graph discovery + multiple RCA algorithms (Bayesian, random walk, etc.) [21] . Domain knowledge can be added [20] .	High – Leverages causal graph; outputs root cause scores [2] . You can see *which* metric and why (based on graph relations).	Local (Python library).	Moderate – Requires defining/ learning causal graph, but well-documented. Integrates with pandas. Has visualization tools for insight.

Solution / Approach	Methodology	Features	Interpretability	Local Deployment	Ease of Integration
ProRCA	Structural causal modeling RCA	Detects anomaly (via ADTK) + finds multi-hop causal pathways by extending DoWhy [26] [30]. Emphasizes multi-factor causes.	High – Provides explicit cause pathways (e.g., A→B→C) with importance scores [28]. Built on causal theory, so results are explainable.	Local (Python library).	Moderate – New library, but uses familiar concepts (pandas, NetworkX graphs). Some coding to prepare data/graph, then one function call to analyze.
DoWhy/ PyWhy	Causal inference analysis	General toolkit for causal effect estimation [33]. Can identify which factor has the largest causal effect on anomaly. Very flexible (not anomaly-specific).	High – Grounded in causal models; you get e.g. "changing X causes Y to change by Z". Requires expert setup (the causal model explains the result).	Local (Python library).	Steeper learning curve – you must manually define models and interpret output. Not a plug-and-play RCA, but very powerful for those familiar with causal analysis.
Elastic Stack ML	Statistical ML in Elastic (X-Pack)	Unsupervised anomaly detection on metrics [41]; log anomaly detection & correlation for RCA [42] [43]. Prebuilt UIs for analysis.	Medium – Anomalies are explained with related metrics or log patterns (the "black box" modeling is hidden, but it points to concrete evidence in data as root cause).	Local with Elastic; requires running Elastic + Kibana.	Setup overhead is non-trivial. Once running, UI-driven configuration is easy. Python integration via API if needed.

Solution / Approach	Methodology	Features	Interpretability	Local Deployment	Ease of Integration
Chaos Genius *(archived)*	ML analytics platform	Offered anomaly detection (Prophet, etc.) and automated RCA drill-downs on metric dimensions [49]. Web interface.	Medium-High – Would show which segments of data caused metric change (e.g., a certain tag or dimension). Less applicable to our use-case without dimensional data.	Local (Docker container).	Moderate – Was relatively easy to deploy and use via UI. Integration with Python not direct (it's its own service). Not maintained now.

Table: Comparison of various root cause analysis solutions for anomalies, focusing on their approach, interpretability, and ease of local integration.

Conclusion and Recommendation

For a **local MacBook deployment in a Streamlit app**, a combination of a lightweight anomaly detector and a causal root cause analyzer is ideal. A practical path forward is:

- **Use a simple anomaly detection first** – e.g., implement threshold checks for ambient temperature limits (for immediate, obvious flags) along with an ML-based detector like Isolation Forest (via PyCaret or PyOD) to catch any subtle anomalies within the "normal" range. This ensures you detect all potentially problematic events with minimal false negatives. The anomaly detection step can run quickly in Python whenever the user selects a time range.

- **Then apply a root cause analysis** using a tool like **PyRCA** or **ProRCA**. Given the context (data center with known factors), you can define a causal graph: for instance, *Door Open → Ambient Temp; External Temp → Ambient Temp; AC Status → Ambient Temp; IT Load → Ambient Temp*. Feed the anomalous period data into PyRCA/ProRCA to get a ranked list of causes. These frameworks will provide a confidence score and you can retrieve an explanation (for example, PyRCA might indicate the AC metric was also anomalous at that time and is upstream of ambient temp in the causal graph, hence it's the top cause [2]).

This hybrid approach leverages the strengths of each component: **high recall anomaly detection** and **accurate, explainable causation analysis**. It remains entirely local (all computations in Python) and transparent (you can log intermediate steps, examine which rules or features fired, and inspect the causal graph reasoning).

If one prefers a more code-light approach and is willing to run extra services, **Elastic Stack's ML features** are a reliable alternative – they encapsulate a lot of the above logic and have been proven in IT operations monitoring. However, integrating Elastic with a Streamlit front-end would involve using its APIs and maintaining a running Elastic instance, which may be overkill for a self-contained app.

In summary, the **most reliable and easy-to-implement solution** is to integrate a Python RCA library (PyRCA or ProRCA) with your anomaly detection. These libraries were **specifically built for IT ops root cause analysis** and will run on a Mac. PyRCA, for example, offers a one-stop solution with anomaly detection, causal graph building, and root cause scoring [17] [21]. ProRCA provides cutting-edge causal analysis with succinct output. Both can output clear cause rankings with confidence metrics, which you can then present in Streamlit as "Root Cause: AC unit off (95% confidence) – likely the AC was not functioning, causing temperature rise" and so on. Combining such a tool with basic rule filters will give you a robust, interpretable RCA feature for data center monitoring.

Lastly, whichever solution you choose, be sure to **validate it on historical incidents**. Fine-tune any thresholds (e.g., what constitutes an external temperature "spike") and update the causal model as you gather more domain knowledge. This will ensure the RCA system remains accurate and trustworthy for the long run [50]. By following these practices and using the tools discussed, you can achieve automated root cause analysis of temperature anomalies that runs entirely on your MacBook and integrates smoothly into your Python/Streamlit application.

Sources: The information and recommendations above are based on a survey of open-source RCA libraries and best-practice guides in anomaly detection and AIOps, including Salesforce's PyRCA [21] [2], a recent causality-based RCA framework (ProRCA) [26] [30], PyCaret's documentation [13], Elastic's observability features [42] [41], and industry case studies on hybrid ML+causal approaches [7]. These sources highlight the importance of combining detection with domain knowledge for effective RCA, and they showcase the viability of running such solutions on local infrastructure.

---

[1] [4] [5] [6] [9] Ultimate Guide to Anomaly Detection and RCA
https://www.eyer.ai/blog/ultimate-guide-to-anomaly-detection-and-rca/

[2] [16] [17] [18] [19] [20] [23] [24] [25] PyRCA: Making Root Cause Analysis Easy in AIOps - Salesforce
https://www.salesforce.com/blog/pyrca/

[3] [10] Network observability: Optimized anomaly detection with AI/ML
https://www.redhat.com/en/blog/network-observability-optimized-anomaly-detection-aiml

[7] [50] Detecting Data Center Anomaly With Machine Learning and Root Cause Analysis - Omdena | Projects | Omdena
https://www.omdena.com/projects/root-cause-analysis

[8] Anomaly Detection Toolkit (ADTK) — ADTK 0.6.2 documentation
https://adtk.readthedocs.io/en/stable/

[11] PyCaret Anomaly Detection Tutorial - Colab - Google
https://colab.research.google.com/github/pycaret/pycaret/blob/master/tutorials/Tutorial%20-%20Anomaly%20Detection.ipynb

[12] Anomaly Detection Tutorial Level Beginner - ANO101 - PyCaret
http://www.pycaret.org/tutorials/html/ANO101.html

[13] What does the Anomaly_Score in Pycaret.anomaly library mean?
https://stackoverflow.com/questions/68746040/what-does-the-anomaly-score-in-pycaret-anomaly-library-mean

[14] Merlion: A Machine Learning Framework for Time Series Intelligence
https://github.com/salesforce/Merlion

[15] In 2024 which library is best for time series forecasting and anomaly ...
https://www.reddit.com/r/MachineLearning/comments/1bho0r0/in_2024_which_library_is_best_for_time_series/

21 22 GitHub - salesforce/PyRCA: PyRCA: A Python Machine Learning Library for Root Cause Analysis
https://github.com/salesforce/PyRCA

26 27 28 29 30 31 32 profitops-rca · PyPI
https://pypi.org/project/profitops-rca/

33 34 35 36 37 38 39 40 Root Cause Analysis with DoWhy, an Open Source Python Library for Causal Machine Learning | AWS Open Source Blog
https://aws.amazon.com/blogs/opensource/root-cause-analysis-with-dowhy-an-open-source-python-library-for-causal-machine-learning/

41 42 43 44 Root cause analysis with logs: Elastic Observability's anomaly detection and log categorization — Elastic Observability Labs
https://www.elastic.co/observability-labs/blog/reduce-mttd-ml-machine-learning-observability

45 46 47 48 49 GitHub - chaos-genius/chaos_genius: ML powered analytics engine for outlier detection and root cause analysis.
https://github.com/chaos-genius/chaos_genius