Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Understanding Service Reliability of Large Language Models: An Empirical Characterization on Operator and User Reports

Author: Yiren Bai (2797993)

1st supervisor:	Prof. dr. ir. Alexandru Iosup
daily supervisor:	MSc. Xiaoyu Chu
2nd reader:	Prof. dr. ir. Tiziano De Matteis

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

October 1, 2025

Abstract

Large Language Model (LLM) services, such as ChatGPT, Claude, and DeepSeek-R1, are increasingly embedded in daily workflows and applications. As their usage scales, ensuring service reliability has become a critical challenge. Existing research predominantly relies on official incident reports, which often lack granularity and overlook user-centered perspectives. This study addresses this gap by systematically analyzing both operator- and user-reported failure data to evaluate LLM service reliability from multiple angles. A structured data pipeline is developed to collect, preprocess, and semantically classify failures from official status pages and third-party reporting platforms. After transforming the collected datasets into a unified data format, the analysis explores failure-recovery characteristics, temporal patterns, cross-source correlations, and reporting consistency across four prominent LLM providers. A total of 28 observations are derived, including findings such as: (1) ChatGPT recovers the slowest from failures, as it has the highest median recovery duration in both the operator and user reports. DeepSeek has the longest median failure interval on the operator side, while Claude shows the longest on the user side; (2) DeepSeek exhibits the largest discrepancy between operator and user reports, in terms of both median recovery durations and median failure intervals; (3) both operator- and user-reported failures display clear periodic and diurnal patterns across services, occurring more frequently on weekdays than on weekends and peaking during the working hours of each service's primary user region; and (4) ChatGPT and Claude show relatively strong alignment between operator and user reports in terms of failure periods, though for different reasons: ChatGPT due to high operator coverage, and Claude due to accurate user reporting and closely aligned user detection. These insights demonstrate the value of integrating heterogeneous failure signals for a more comprehensive understanding and evaluation of reliability, and provide a foundation for future work on predictive failure modeling and user-aware, real-time reliability monitoring systems.

Contents

List of Figures

List of Tables

1 Introduction

LLMs, such as OpenAI's ChatGPT [29], Anthropic's Claude [7], and DeepSeek-R1 [14], have rapidly become foundational technologies powering a broad spectrum of applications—from conversational assistants and coding support tools to enterprise automation and decision-making systems [33, 18]. The widespread integration of LLMs into mission-critical tasks has substantially heightened the importance of ensuring their operational reliability. Unlike many traditional cloud services, where failure states are typically captured using binary availability metrics such as uptime or downtime [25], LLM services tend to exhibit more complex and less easily quantifiable failure behaviors [35]. While partial degradations and performance variability are not unique to LLMs, they become significantly more pronounced due to the model-dependent nature of LLM workloads and the multi-layered infrastructure on which these systems rely. Such failures often involve intermittent disruptions, functional inconsistencies—such as semantically incorrect outputs or degraded feature modules despite nominal service availability—or delayed responses, all accompanied by highly unpredictable recovery durations [26]. As a result, they can lead to subtle yet substantial impacts on both user experience and operational reliability.

However, existing research often falls short in addressing these complexities. First, current studies primarily rely on operator reports [12], which offer a provider-side view of system status but lack access to large-scale, structured user reports. This limits the ability to understand how failures manifest from the user perspective. Second, there is a lack of effective tools and methodologies to systematically collect, characterize, and compare operator and user reports. This leads to fragmented data sources and hinders cross-perspective reliability analysis. Third, the current understanding of LLM service failures remains limited [37]. Many studies focus on coarse-grained metrics such as uptime or error rates [38], while overlooking subtler forms of degradation, including response-quality issues such as hallucinations [22], transient latency spikes, and inconsistencies in feature-level functionality. These issues often vary in scope and impact across different user groups, making them difficult to detect and quantify using operator-side data alone.

To address these limitations, this study conducts a multi-perspective operational analysis of LLM service reliability. The analysis is based on a custom-built data collection and processing pipeline that gathers operator and user reports from four major LLM providers, including OpenAI, Anthropic, and DeepSeek, as well as from three third-party reporting platforms. The resulting datasets include 530 operator reports spanning 18 months and over 160,000 user reports collected over a 3-month period. Based on these datasets, the study performs several types of analysis, including failure-recovery modeling, temporal pattern analysis, correlation analysis, and consistency analysis between the two sources. The results reveal three key findings. First, ChatGPT has the slowest recovery from failures, with both operator and user reports showing the highest median recovery time. In terms of failure intervals, DeepSeek exhibits the longest median interval on the operator side, while Claude shows the longest on the user side. Second, failures reported by both operators and users exhibit clear periodic and diurnal patterns across services, occurring more frequently on weekdays and peaking during the working hours of each service's primary user region. Third, ChatGPT and Claude demonstrate relatively strong alignment between operator and user reports. These findings offer broader insights into the reliability characteristics of LLM services.

This study offers significant practical implications for industry, academia, and society. For service providers, the findings support improved incident detection, enabling more effective and timely responses and fostering proactive reliability management. From an academic perspective, this work lays the groundwork for future research in predictive reliability modeling, resilience benchmarking, and comparative evaluations of LLM-driven service reliability. For society, particularly for non-technical users, the study provides clearer insights into service disruptions, helping them better understand the behavior and limitations of LLM services. Together, these contributions not only advance the understanding of LLM reliability but also support the broader goal of developing more robust and user-centric LLM services.

1.1 Research Background

The widespread adoption of LLMs has significantly transformed various industries. Unlike traditional software services, LLMs are typically deployed as cloud-based APIs, enabling users to leverage their capabilities without managing the underlying infrastructure. For example, OpenAI's ChatGPT is hosted on Microsoft Azure [40], while Anthropic's Claude [2] and Google's Gemini [32] operate on Google Cloud Platform (GCP). However, this reliance on remote inference introduces new challenges in maintaining service reliability. As these models become integral to critical applications, understanding their operational stability and failure characteristics becomes essential.

LLM Service Reliability and Evaluation Approaches Service reliability is a core concern for cloud-based computing, extending beyond basic uptime metrics such as availability percentages and Service Level Agreements (SLAs) to encompass performance stability, resilience to failures, and rapid recovery. In traditional cloud environments, disruptions primarily result from predictable causes like hardware failures, network outages, or software bugs, which are typically managed using redundancy, failover mechanisms, and load balancing [16].

LLM services introduce unique challenges not encountered in conventional cloud systems. Due to real-time model inference requirements, each user query consumes significant computational resources dynamically, resulting in service performance fluctuations even under stable infrastructure conditions [24]. Unlike deterministic traditional services, the dynamic and probabilistic nature of LLM inference often results in disruptions that manifest as degraded response quality, increased latency, or intermittent access failures, rather than outright service outages. Additionally, external dependencies—such as cloud storage solutions, vector databases, and model-serving infrastructures—complicate reliability management. Resource allocation optimizations designed to meet real-time demand can inadvertently trigger transient disruptions during rapid scaling events or traffic surges. Consequently, LLM service reliability is shaped not only by infrastructure stability, but also by inference optimization strategies, model architecture decisions, and resource allocation constraints.

Evaluating LLM service reliability remains a complex task, as industry practices continue to evolve without standardized methodologies. Most service providers define SLAs primarily in terms of availability metrics, while aspects such as degraded response quality or intermittent disruptions are often overlooked. Major LLM providers, including Ope-

nAI, Anthropic, and Google DeepMind, employ internal observability pipelines to monitor infrastructure health, API request patterns, and system performance metrics. These systems typically include real-time dashboards and automated alerts to facilitate rapid detection and mitigation of service degradations. However, such internal tools are generally inaccessible to external users, limiting independent evaluations of service reliability. To complement provider-side monitoring, third-party platforms such as DownDetector and DownForEveryoneOrJustMe aggregate real-time user reports, offering valuable insights into externally observable service failures.

Additionally, academic and industry researchers have developed synthetic benchmarking frameworks to systematically evaluate the reliability of LLM services. These frameworks typically issue standardized queries to LLMs and record metrics such as response latency and error rates [38]. While benchmarking offers structured and quantitative insights, its effectiveness is often constrained by provider-imposed rate limits, which can introduce sampling bias and limit reproducibility.

Although each of the aforementioned approaches contributes valuable perspectives, their fragmentation underscores the need for more comprehensive and multi-perspective methodologies—particularly those that leverage both operator and user reports—to enable holistic evaluations of service reliability.

LLM Service Failure Types LLM service failures exhibit diverse characteristics that distinguish them from those of conventional cloud-based applications. Rather than manifesting solely as complete service outages, these failures often involve complex, multi-stage degradation patterns. Drawing on recent empirical studies [35], LLM service failures can be broadly grouped into the following categories:

Authentication failures prevent users from logging into LLM platforms or accessing API services. These failures often manifest as *Login Failed* or *Invalid Credentials* errors. They may result from authentication system outages, rate-limiting mechanisms that mistakenly block legitimate access, or unexpected changes in authentication policies. Since authentication services are often decoupled from model inference, such failures may occur even when the core LLM remains functional.

Request errors occur when users submit queries to an LLM service but receive error responses such as HTTP 500 (Internal Server Error), 502 (Bad Gateway), 503 (Service Unavailable), or 504 (Gateway Timeout). These failures typically result from backend server overload, misconfigured routing policies, or crashes in the model-serving infrastructure. While transient failures may be resolved through automatic retries, persistent failures often indicate deeper stability issues within the service architecture.

High response latency is another common failure type, in which response times significantly exceed normal levels. Users may experience prolonged wait times or even timeouts without receiving a response. Latency spikes often stem from resource contention, inefficient model inference pipelines, or increased computational overhead caused by dynamic scaling strategies. Unlike outright service failures, latency-related issues degrade the user experience in subtler ways and may not always be reflected in operator reports.

Partial feature failures occur when certain functionalities within an LLM service become unavailable, even though the core model remains operational. For example, multimodal capabilities, fine-tuned models, or API-integrated plugins may cease functioning due to isolated microservice outages or deployment inconsistencies. These failures disproportion-

ately affect users who rely on specific features, leading to fragmented perceptions of service reliability across different customer segments.

Major outages represent large-scale service failures in which all API endpoints and user interfaces become inaccessible. These failures may stem from cloud infrastructure disruptions, catastrophic software bugs, or breakdowns in distributed model-serving clusters. Unlike partial feature failures, major outages typically affect all users simultaneously and often trigger official incident reports from service providers.

The failure-recovery process for LLM services typically follows a structured sequence that includes investigation, issue identification, repair implementation, post-repair monitoring, and root cause analysis [12]. However, due to the probabilistic nature of model inference and the reliance on distributed cloud infrastructure, recovery times can be unpredictable, and failures may recur under high-load conditions. As LLM adoption continues to expand, understanding these failure types is essential for improving service resilience and ensuring stable user experiences.

1.2 Problem Statement

While understanding and evaluating LLM service reliability from multiple perspectives has become essential, significant challenges persist in both data collection and analysis, limiting meaningful insights into failure patterns and service stability. This study aims to address two key challenges: the lack of comprehensive, structured failure data and the absence of standardized analytical approaches tailored to assessing service reliability.

Lack of Comprehensive Failure Data Evaluating LLM service reliability requires a unified data format that makes operator and user reports comparable, enabling comprehensive reliability assessments. Currently, failure data is fragmented across multiple sources, each presenting distinct challenges for integration and consistency:

Operator reports sometimes use inconsistent terminology across providers and generally lack sufficient granularity. For example, two providers may describe the same type of failure in different ways—one using a vague label such as *Degraded performance*, while another offers a more specific description like *Elevated latency*—creating ambiguity. Furthermore, providers typically aggregate incident reports across services, subscription tiers, and geographic regions, obscuring insights into localized issues. For instance, an outage affecting only specific components or users in particular geographic areas may be inadequately reflected in broadly summarized reports.

User reports from third-party platforms like DownDetector present a different set of challenges. These reports are often unstructured, brief, or ambiguous—for example, vague descriptions such as *ChatGPT is down* or *not working* without context. Extracting structured and accurate failure types from such free-text data requires the semantic capabilities of LLMs; basic natural language processing (NLP) techniques are typically insufficient. Additionally, historical user-reported failures are often unavailable due to the data retention policies of reporting platforms such as DownForEveryoneOrJustMe, which typically preserve only a fixed number of user reports—20 in this case—making longitudinal reliability studies difficult, as the data cannot be reproduced.

Furthermore, collecting user reports introduces complexities related to false positives, where users may mistakenly attribute personal connectivity issues or localized network

problems to LLM service failures. For example, complaints on social media about perceived disruptions may actually stem from regional network conditions or client-side issues, making it difficult to accurately identify genuine service failures. Such attribution errors introduce noise into the dataset and complicate the reliable detection and classification of actual disruptions.

Absence of Standardized Analytical Approaches Existing research [12] on LLM service reliability primarily relies on operator reports published on official status pages. While these reports provide valuable insights from the provider's perspective—such as incident timelines, affected components, and high-level failure causes—they often fail to reflect the finer-grained, symptom-level disruptions experienced by end users. As discussed earlier, operator reports frequently lack detail and geographic specificity, making them inadequate for capturing the full scope of service behavior as experienced in real-world usage contexts. Consequently, evaluations based solely on operator data offer a limited and potentially skewed view of service reliability.

A more comprehensive understanding requires analyzing service disruptions from both operator and user perspectives in parallel. This, in turn, necessitates standardized analytical approaches that can effectively accommodate heterogeneous data sources, align their formats and semantics, and enable meaningful cross-source comparisons. However, such methodologies remain largely absent in current literature [8]. For example, analyses that evaluate the degree of alignment between overlapping failure periods in operator and user reports, as well as identify patterns such as whether user-perceived failures systematically precede official acknowledgments, are rarely conducted. Without such multi-perspective evaluations, critical dimensions of reliability—such as service transparency, responsiveness, and user-perceived impact—remain insufficiently addressed. This study seeks to bridge that gap by introducing a set of comparative analytical approaches that incorporate both operator and user reports, enabling more robust and multi-perspective evaluations of LLM service reliability.

1.3 Research Questions

This research aims to provide a comprehensive operational analysis of LLM service reliability by leveraging multi-source failure data from both service providers and users. Our main question is: **How to collect and understand LLM service reliability through different sources of failure data?** This main research question can be divided into the following sub-questions (RQ):

RQ1: How to collect, process, and unify operator- and user-reported failure data to support reliability analysis of LLM services?
This question aims to develop a standardized data pipeline capable of harmonizing heterogeneous reports across time zones, formats, and semantics. It involves data scraping, timestamp normalization, semantic classification of failure types, and failure period extraction to build structured datasets for analysis. (Section 2)

RQ2: How to characterize failure recovery patterns across LLM services?
This question focuses on how failures manifest and evolve by analyzing recovery durations, failure intervals, failure frequencies, and other related aspects through reliability metrics such as MTTR and MTBF. The goal is to model failure behaviors and to evaluate and compare service robustness from both operator and user perspectives. (Section 3)

RQ3: What temporal patterns emerge in LLM service failures, and how can they inform proactive reliability strategies?
This question explores whether service failures exhibit temporal clustering or follow regular cycles across hourly, daily, or weekly intervals. To uncover such patterns, the analysis applies methods including trend visualization, categorical time-based distributions, and autocorrelation analysis. Identifying these temporal characteristics can provide valuable insights for predicting failures and designing proactive reliability strategies. (Section 4)

RQ4: How strongly correlated are operator and user-reported failure signals, and what inter-service dependencies can the analysis identify?
This question examines the statistical relationship between operator-reported incidents and user-reported failure volumes, as well as the co-occurrence of failures across different services. It seeks to uncover potential operational entanglements or shared infrastructure vulnerabilities and to evaluate whether user reports accurately reflect or even anticipate official disclosures. (Section 5)

RQ5: To what extent do operator reports align with user experiences, and what does this reveal about provider reporting practices?
This question investigates the consistency between operator and user reports across two key dimensions: failure types and failure periods. The analysis involves computing match rates, generating heatmaps of type-to-type mappings, and comparing reported timelines to identify both alignment and divergence patterns. These findings help evaluate the coverage and granularity of provider reporting practices. (Section 6)

1.4 Research Contributions

Our study provides five research contributions (RC), including:

RC1 (Conceptual): We propose a data collection and analysis methodology for understanding LLM service reliability through multiple sources of failure reports. As part of this methodology, we design a unified data processing pipeline that standardizes and structures heterogeneous raw inputs, such as official incident records, user comments, and social media posts, into analysis-ready datasets. The processing steps include deduplication, time zone normalization, failure type classification, and extraction of user-reported failure periods. By transforming diverse and unstructured inputs into a unified format, the methodology enables consistent cross-source comparisons and supports a more comprehensive, user-centric evaluation of service reliability from both operator and user perspectives.

RC2 (Technical): We implement a data collection and analysis tool for processing and characterizing LLM failure data. Using this tool, we collect 530 operator reports from four major providers over an 18-month period, along with over 160,000 user reports from three third-party platforms spanning 3 months. After data cleaning, we apply the semantic reasoning capabilities of LLMs in a few-shot setting to classify failure types from unstructured incident descriptions and user comments. We also aggregate user reports into 1-minute bins and apply a 7-minute moving average to infer failure periods. These steps transform the raw inputs into a structured and unified format for downstream failure analysis.

RC3 (Conceptual): We propose four types of failure analysis on our collected datasets: failure-recovery modeling, temporal pattern analysis, correlation analysis, and consistency analysis. Each type targets a distinct dimension of service reliability. Failure-recovery modeling quantifies robustness using metrics such as recovery duration, failure interval, failure frequency, and availability percentage. Temporal analysis explores periodic and diurnal failure patterns using categorical time-based distributions and autocorrelation functions. Correlation analysis captures failure co-occurrence across services using a conditional probability matrix. Finally, consistency analysis evaluates the alignment between operator and user reports in both semantics and timing. Semantic consistency is measured by failure type match rates, while temporal consistency is evaluated using metrics such as the percentage of overlapping reports and the lead time of user-reported failures relative to operator reports. This comprehensive analytical approach enables a multi-faceted understanding of LLM service reliability.

RC4 (Technical): We conduct 11 types of analysis and summarize 28 important observations to provide insights based on long-term failure data from prominent LLM services. We find that: (1) ChatGPT recovers the slowest from failures, with the highest median recovery duration reported by both the operator (1.07 hours) and the user (1.23 hours). DeepSeek has the longest median failure interval on the operator side (5.39 days), while Claude shows the longest on the user side (1.07 days); (2) DeepSeek exhibits the largest discrepancy between operator and user reports, in terms of both median recovery durations (0.81-hour gap) and median failure intervals (4.6-day gap); (3) failures reported by both operators and users display clear periodic and diurnal patterns across services,

occurring more frequently on weekdays and peaking during the working hours of each service's primary user region; and (4) ChatGPT and Claude show relatively strong alignment between operator and user reports in terms of failure periods, though for different reasons: ChatGPT has a high operator coverage of 82.22%, while Claude shows accurate user reporting at 74.00%, along with closely aligned user detection, with a median user lead time of 3 minutes.

RC5 (Open Science): We will release the collected datasets and the relevant toolkit as a contribution to open science.

1.5 Research Structure

The remainder of this study is structured as follows. Section 2 introduces the data sources used in this work and outlines the data collection pipeline, processing steps, and four types of analytical approaches. Section 3 presents a failure-recovery modeling approach to characterize failure dynamics. Section 4 conducts temporal trend and distribution analyses and applies auto-correlation techniques to uncover service-specific patterns in failure occurrences. Section 5 examines the correlation between operator-reported timelines and user-reported issue volumes, and analyzes co-occurring failures across services to identify potential interdependencies. Section 6 presents a consistency analysis to evaluate the alignment between operator and user reports in terms of failure types and reported periods. Section 7 discusses threats to the validity of the study. Section 8 reviews related work, and Section 9 concludes the study.

2 Method for Analyzing Operator and User Reports

The approach proposed in this research systematically addresses the identified challenges by establishing a comprehensive data pipeline and analytical workflow tailored specifically to LLM services. This pipeline integrates multiple data sources—including official operator reports and user reports—and implements structured data collection, processing, and analytical techniques designed to provide granular and actionable insights into the reliability of LLM services.

2.1 Data Collection

This data pipeline, as illustrated in Fig. 2.1, collects both official operator reports and user reports from third-party reporting platforms (e.g., DownDetector and DownForEveryoneOrJustMe) as well as social media (e.g., Twitter/X). It covers four major providers and their respective web-based chat services: OpenAI's ChatGPT, Anthropic's Claude, DeepSeek's DeepSeek, and Character.AI.

The API services of these LLM providers are not monitored in this study. This is because there are insufficient user-reported failures to support comparative analysis. Most API users are business clients, who typically reach out to customer support directly rather than report issues through public platforms such as DownDetector or social media. As a result, user-reported failure data for API services is sparse or unavailable. In contrast, the web-based chat services of these LLM providers have significantly larger user bases, making it much more likely that user-reported failures will be captured in sufficient quantity for downstream analysis.

Operator Reports Incident reports from official LLM service providers are collected from their status pages (e.g., *status.openai.com*, *status.anthropic.com*). Specialized scraping scripts extract all incident reports, generating two output formats per service: (i) a consolidated CSV file containing summary information about all incidents and (ii) a detailed JSON file preserving the complete incident report content. These reports contain important information about failures, including impact levels (*Full Outage, Partial Outage, Degraded Performance, and Maintenance*), failure start times, recovery times, failure types, and failure root causes (if available). All collected data is initially stored locally before further processing.

User Reports To incorporate user perspectives on LLM service failures, three parallel data collection streams have been developed:

- **DownDetector**: Two separate scraping solutions were implemented—one extracts user comments from service-specific pages, and another collects social media reports aggregated by DownDetector. These scripts run on a daily scheduled basis through Airflow orchestrated in Docker containers, performing incremental data collection. The collected data is stored in an AWS S3 bucket.
- **DownForEveryoneOrJustMe**: This platform presents unique challenges, as it only displays the 20 most recent user reports at any given time. To capture historical data, a persistent monitoring solution was deployed on a GCP e2-micro instance,

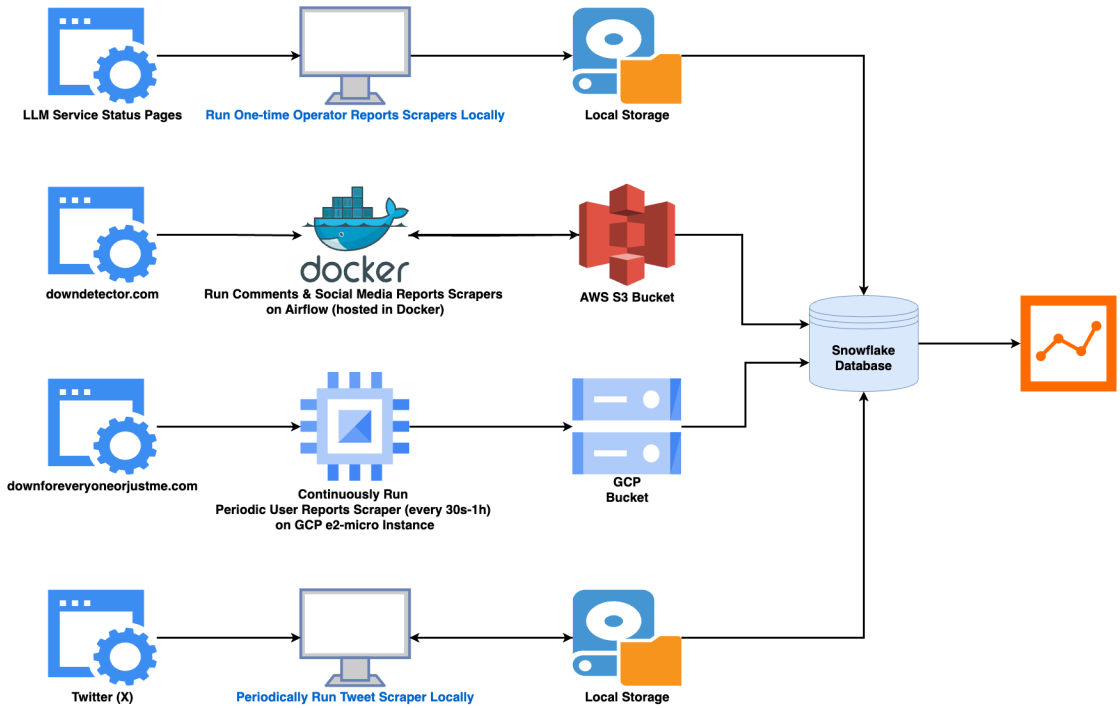


Figure 2.1: Data pipeline for collecting operational reports of LLM services, integrating both official operator reports and user reports from third-party reporting platforms and social media to enable comprehensive reliability analysis.

using screen sessions to ensure continuous operation. The collection frequency dynamically adjusts between 30 seconds and 1 hour, based on observed update patterns for each LLM service. All incremental data is stored in a dedicated GCP bucket.

- **Twitter/X**: While DownDetector includes some social media reports, it sometimes fails to update in a timely manner, leading to fragmented updates. Therefore, a dedicated Twitter scraping solution was developed to supplement this data. Due to rate limitations and authentication requirements, this script requires periodic manual execution with credential input. The script extracts incremental tweet data related to LLM service experiences and stores it locally in service-specific files.

Dataset Overview After approximately three months of data collection, all datasets have been aggregated into a Snowflake database for further processing and analysis. The directory tree structure of these datasets is illustrated in Fig. 2.2. As the figure indicates, the archive of operational reports for LLM services consists of six types of datasets.

For operator reports, two types of datasets were collected. One provides summary-level information on incident reports for each LLM service, presented in Table 2.1. The other offers detailed records for each individual incident report, shown in Table 2.2.

For user reports, the first source is DownDetector, which contributes two datasets: one containing user comments about each LLM service, displayed in Table 2.3, and another capturing user reports from social media (i.e., tweets), outlined in Table 2.4. The second

source is DownForEveryoneOrJustMe, where each LLM service has a dedicated dataset of user reports, provided in Table 2.5. The third source is Twitter/X, which also yields a separate dataset of tweets for each LLM service, summarized in Table 2.6. The dataset scraped directly from Twitter/X primarily serves as a supplementary source to the DownDetector tweets dataset. Given the occasional omissions in DownDetector's updates of social media reports, manual scraping from Twitter/X is used as a fallback when necessary.

Table 2.7 presents a consolidated summary of the collected datasets, including both operator and user reports.

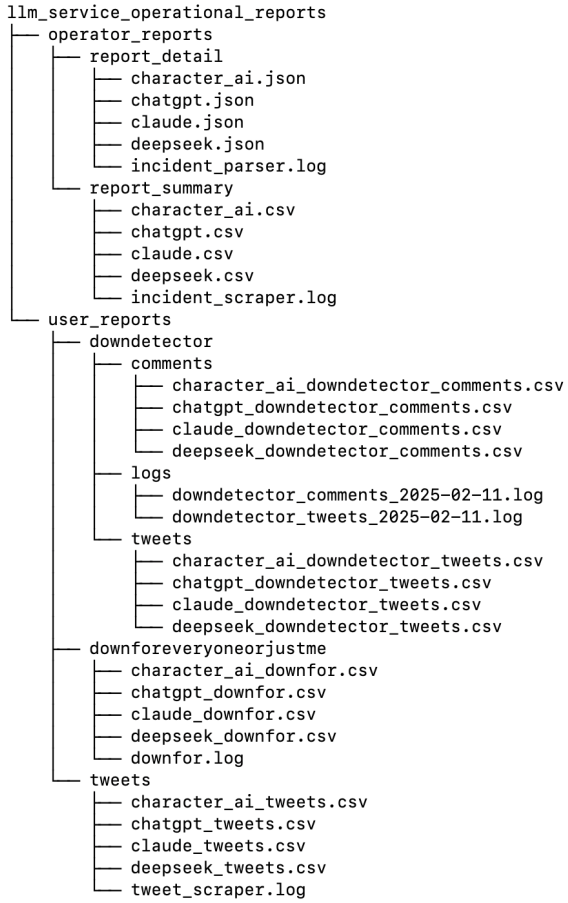


Figure 2.2: Directory structure of LLM service operational reports.

Table 2.1: Fields and example entries from the operator report summary dataset.

Field	Example Entry
title	Elevated errors on Claude.ai and the Anthropic Console
url	https://status.anthropic.com/incidents/rv4qtngkcdny
impact_level	major
description	This incident has been resolved.
start_time	Mar 31, 2025 - 07:52 PDT
end_time	Mar 31, 2025 - 09:18 PDT

Table 2.2: Fields and example entries from the operator report detail dataset.

Field	Example Entry
basic_info.title	Elevated errors on Claude.ai and the Anthropic Console
basic_info.impact_level	major
affected_components	claude.ai, console.anthropic.com, api.anthropic.com
timeline[0].status	Resolved
timeline[0].content	This incident has been resolved.
timeline[0].timestamp	Posted Mar 31, 2025 - 09:18 PDT
postmortem	null
source_url	https://status.anthropic.com/incidents/rv4qtngkcdny
source_file	claude

Table 2.3: Fields and example entries from the DownDetector comments dataset.

Field	Example Entry
username	Daniel Holida
datetime_string	Monday, March 31, 2025 1:44 PM
timestamp	2025-03-31 13:44:00
content	Oh god please don't make me actually search for an answer on stackoverflow
likes	1

Table 2.4: Fields and example entries from the DownDetector tweets dataset.

Field	Example Entry
username	@rchitectopteryx
timestamp	2025-03-31 14:36:25
content	Every morning, as the west coast. wakes up, ChatGPT becomes unusable.

Table 2.5: Fields and example entries from the DownForEveryoneOrJustMe dataset.

Field	Example Entry
timestamp	2025-03-30T21:15:43.119Z
country	United States
issue_type	Slow

Table 2.6: Fields and example entries from the Twitter/X dataset.

Field	Example Entry
tweet_id	1906758818945437991
text	Smfh ChatGPT is still down
created_at	2025-03-31 17:21:42+00:00
author_id	19307054
author_name	can you lend a DEI a pencil?
author_username	RodSteed
retweet_count	0
reply_count	0
like_count	1
lang	en

Table 2.7: Summary of collected datasets. Legend: D=day(s).

Platform	Dataset Content	Service	First Date	Last Date	Coverage [D]	# of Reports
Status Pages	Incident Summary	ChatGPT	2023-12-01	2025-05-18	535	179
		Claude	2023-12-01	2025-05-18	535	227
		DeepSeek	2024-05-01	2025-05-18	383	37
		Character.AI	2023-12-01	2025-05-18	535	87
	Incident Detail	ChatGPT	2023-12-01	2025-05-18	535	179
		Claude	2023-12-01	2025-05-18	535	227
		DeepSeek	2024-05-01	2025-05-18	383	37
		Character.AI	2023-12-01	2025-05-18	535	87
DownDetector	Comments	ChatGPT	2025-01-02	2025-05-18	137	2,551
		Claude	2025-01-30	2025-05-18	109	18
		DeepSeek	2025-01-29	2025-05-18	110	57
		Character.AI	2025-01-04	2025-05-18	135	12,237
	Social Media Reports	ChatGPT	2025-02-11	2025-05-18	97	2,966
		Claude	2024-12-29	2025-05-18	141	1,006
		DeepSeek	2025-02-18	2025-05-18	90	406
		Character.AI	2024-10-24	2025-05-18	207	47
DownForEveryoneOrJustMe	User-Reported Issues	ChatGPT	2025-02-13	2025-05-18	95	38,160
		Claude	2025-02-06	2025-05-18	102	1,900
		DeepSeek	2025-02-12	2025-05-18	96	1,377
		Character.AI	2025-02-13	2025-05-18	95	73,285
Twitter/X	Tweets	ChatGPT	2025-01-01	2025-05-18	138	10,740
		Claude	2025-01-01	2025-05-18	138	8,205
		DeepSeek	2025-01-01	2025-05-18	138	6,000
		Character.AI	2025-01-01	2025-05-18	138	5,461

2.2 Data Process

To enable structured analysis, the collected data is processed and transformed into a unified format. Depending on the source and structure of each dataset, different processing strategies are applied. The details are outlined below.

2.2.1 Data Cleaning and Failure Type Extraction

Operator Report Datasets For the operator report summary dataset, timestamps such as `start_time` and `end_time` are recorded in different time zones depending on the specific provider. To ensure consistency across all datasets, these timestamps are converted to Coordinated Universal Time (UTC), thereby establishing a unified temporal format for downstream analysis.

The main task in processing the operator report detail dataset is to classify each incident into a predefined failure type using an LLM in a few-shot setting. This approach leverages the semantic reasoning capabilities of LLMs, enabling more accurate and context-aware classification than traditional rule-based or keyword-based methods.

For each incident report, three key textual elements are extracted and concatenated to form the classification input: the `title` subfield from the `basic_info` field, and all `content` subfields from the `timeline` field that describe the incident's progression. The combined text captures all the necessary information about a given incident, such as symptoms, root cause, and remediation efforts.

This input is then passed to the LLM (i.e., ChatGPT-4o), which leverages its semantic capabilities to infer the most appropriate failure type from the overall context[1]. The predefined classification categories align with those discussed in Section 1.1 and follow the logical sequence of a typical user interaction flow—from initial access to actual usage. The categories are as follows:

- **Login**: Users encounter issues while attempting to log in or sign up.
- **Inaccessible**: The entire service or specific features become unavailable.
- **Slow**: The service exhibits elevated latency or significantly degraded response times.
- **Error Received**: Users encounter explicit errors during usage, such as system messages (e.g., *Something went wrong*) or technical error codes (e.g., HTTP 5xx).
- **Others**: Issues that do not clearly fall into any of the above categories, including scheduled maintenance.
- **Unknown**: The operator report does not specify the type of issue.

After performing the classification task, the LLM generates an output dataset in CSV format, consisting of three columns: `url`, `affected_components`, and `failure_type`.

Once both operator report datasets have been processed, the cleaned summary dataset is merged with the aforementioned three-column output dataset using the `url` field as the key. This produces a unified dataset, the structure of which is shown in Table 2.8.

[1]See the Appendix for the full prompt used in the structured extraction process.

Table 2.8: Fields and example entries from the final operator report dataset.

Field	Example Entry
title	Elevated errors for ChatGPT
affected_components	ChatGPT
impact_level	Degraded performance
start_time	2025-02-19 18:42:00
end_time	2025-02-19 19:08:00
failure_type	Error Received

DownDetector Dataset The DownDetector comments dataset consists of user feedback directly submitted on the DownDetector platform. Each entry in the dataset corresponds to a single user comment regarding a specific LLM service, and includes a textual description (`content`), a timestamp (`timestamp`), and a count of likes received from other users (`likes`). To extract actionable signals from this dataset, as illustrated in Table 2.3, a multi-step data processing pipeline was designed.

First, the `content` column was processed using ChatGPT-4o with a carefully designed few-shot prompt. This step aimed to identify the failure type perceived by the user in each comment. Such classification aligns with that used for operator reports, consisting of six categories: `Login`, `Inaccessible`, `Slow`, `Error Received`, `Others`, and `Unknown`. The predicted failure type for each comment was stored in a new column named `failure_type`.

Next, the `likes` column was incorporated to reflect how many additional users resonated with the reported issue. A new column, `weight`, was introduced to reflect the total number of users affected by the same problem. It is defined as `weight = likes + 1`, where the additional one accounts for the original commenter. This weighting scheme assumes that a higher number of likes indicates broader user agreement with the reported issue.

Combining the original `timestamp` column (already normalized to UTC), the extracted `failure_type`, and the computed `weight`, a processed dataset was constructed to capture temporal and categorical patterns in user-reported service failures. Each row in this dataset, exemplified in Table 2.9, represents a timestamped instance of perceived service failure, along with its inferred failure type and the estimated number of affected users.

Table 2.9: Sample entries from the processed DownDetector comments dataset, showing report timestamps, inferred failure types, and comment weights.

timestamp	failure_type	weight
2025-04-28 18:38:00	Slow	3
2025-04-28 18:40:00	Error Received	1
2025-04-28 19:41:00	Inaccessible	2
2025-04-28 21:10:00	Slow	2
2025-04-29 05:33:00	Error Received	1
2025-04-29 19:17:00	Slow	1
2025-04-29 20:29:00	Error Received	2

To identify periods of user-perceived service failure, the processed data was further

aggregated into one-minute intervals. For each minute, all entries within the corresponding time window were grouped, and their `weight` values summed to compute a new column `report_count`, representing the total number of user reports for that minute. This results in a second-level dataset, illustrated in Table 2.10, consisting of two columns: `timestamp` (minute-level granularity) and `report_count`. This aggregated representation forms the foundation for constructing a unified multi-source time series of user report counts for subsequent analysis.

Table 2.10: Sample time series entries from the processed user report dataset aggregated at one-minute intervals.

timestamp	**report_count**
2025-04-28 18:38:00	3
2025-04-28 18:39:00	0
2025-04-28 18:40:00	1
2025-04-28 18:41:00	0
2025-04-28 18:42:00	0

The DownDetector tweets dataset contains user posts collected by DownDetector from external social media platforms, primarily Twitter/X. Its processing pipeline closely follows that of the comments dataset. As before, the `content` column in the original dataset was processed using ChatGPT-4o with a tailored few-shot prompt, enabling semantic analysis to identify the service failure type described in each post. The inferred failure types were then stored in a new column named `failure_type`.

Unlike the comments dataset, the tweets dataset does not include a `likes` column. As a result, the processed dataset consists only of the original `timestamp` column and the extracted `failure_type` column, as illustrated in Table 2.11.

To construct a second-level dataset reflecting the number of user reports per minute, the processed data was aggregated at one-minute intervals. Since no `weight` values are available, the `report_count` for each interval was computed as the total number of entries within that minute. The resulting dataset shares the same structure as the one shown in Table 2.10, with columns `timestamp` and `report_count`.

Table 2.11: Sample entries from the processed DownDetector tweets dataset with report timestamps and inferred failure types.

timestamp	**failure_type**
2025-05-03 00:31:28	Error Received
2025-05-03 02:30:36	Inaccessible
2025-05-03 02:50:55	Error Received
2025-05-03 04:04:53	Slow
2025-05-03 04:44:16	Others
2025-05-03 06:00:19	Inaccessible
2025-05-03 08:40:11	Error Received

Twitter/X Dataset As a complement to the DownDetector tweets dataset, the Twitter/X dataset follows the exact same processing pipeline as the DownDetector comments dataset. This is because the structure and semantics of the relevant columns are fully aligned: the `text` column in the Twitter/X dataset corresponds to `content`, the `created_at` column corresponds to `timestamp`, and the `like_count` column corresponds to `likes` in the DownDetector comments dataset.

Accordingly, the same steps were applied: first, the `text` column was semantically analyzed using ChatGPT-4o to infer the failure type, stored in a new column `failure_type`; second, the `like_count` values were used to compute a new `weight` column by adding one to each value; and finally, these processed rows were aggregated at one-minute intervals to produce a time series of user report counts. As a result, the processed Twitter/X dataset has its own counterparts of Table 2.9 and Table 2.10, following the same data structure and interpretation.

DownForEveryoneOrJustMe Dataset For the dataset collected from DownForEveryoneOrJustMe, no LLM-based semantic parsing is required, as the original data already includes both the timestamp of each user-reported issue (`timestamp`) and the corresponding failure type (`issue_type`). Therefore, unlike the DownDetector or Twitter/X datasets, the data can be directly aggregated at one-minute intervals to count the number of user reports, resulting in a time series dataset similar in structure to Table 2.10.

Once the raw datasets from the three platforms have each been processed into time series datasets representing user report counts over time, the final step is to merge them. This is done by grouping all records by their `timestamp` and summing the `report_count` values within each group. The result is a unified, minute-level dataset capturing the total number of user reports across all sources, which serves as the foundation for identifying periods of user-perceived service failure.

2.2.2 Inferring User-Reported Failure Periods

Identifying user-reported failure periods from this unified dataset presents a notable challenge due to the highly jagged nature of the original time series of user report counts. With one-minute granularity, the data often alternates between sharp spikes and intervals with zero reports, making it difficult to determine whether a failure is continuous or intermittent. For example, given a detection threshold of 6 reports per minute, it is ambiguous whether a sequence such as 6, 6, 5, 6, 6 should be treated as one continuous failure period or whether the brief drop below the threshold should segment it into multiple episodes.

To address this, a moving average method is applied to smooth the time series, as shown in Fig. 2.3. This method includes two key parameters: the moving average window size and the detection threshold, which defines the minimum report volume required to classify a period as a failure. All time segments in the smoothed curve that remain above the threshold are extracted as user-reported failure periods.

To choose the most appropriate parameter combinations that best reflect the time periods when users actually perceived service failures, a sensitivity analysis is applied. Since no ground truth labels are available for user-reported failure periods, traditional evaluation metrics such as accuracy or recall are not applicable. Instead, this study adopts a fragmentation-based heuristic to evaluate different parameter combinations.

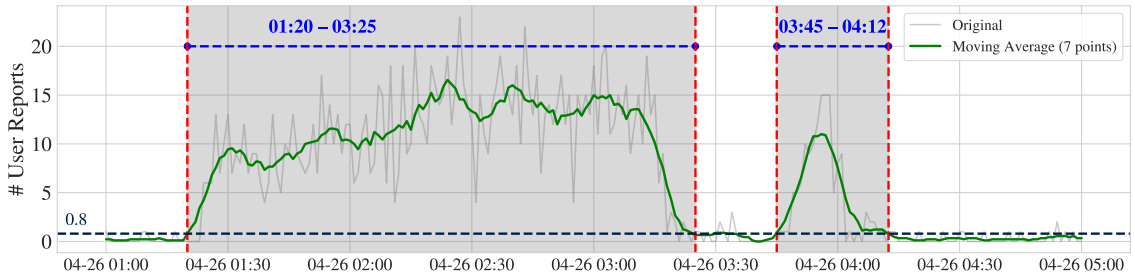


Figure 2.3: Visualization of a segment of user report volume over time for ChatGPT, spanning 01:00 to 05:00 on April 26, 2025. The gray shaded area indicates the period identified as user-reported downtime.

The core idea is that inappropriate parameters may result in fragmented failure segments that should have been detected as a single continuous period. To quantify the degree of fragmentation in a sequence of detected user-reported failure periods, a *fragmentation score* is introduced. Let $P = \{(s_i, e_i)\}_{i=1}^{n}$ denote the set of detected failure periods under a given parameter setting, where s_i and e_i represent the start and end timestamps of the i-th failure period, respectively. A *mergeable pair* is defined as any two consecutive periods (e_i, s_{i+1}) such that the gap between them satisfies:

$$s_{i+1} - e_i \leq \delta \tag{1}$$

where $\delta = 5$ minutes is a predefined merge tolerance. Intuitively, if two failure periods occur within 5 minutes of each other, they can be considered part of a single perceived failure episode that was artificially split due to a short fluctuation in report volume.

Based on this definition, the *fragmentation score* is given by:

$$\text{Fragmentation Score} = \frac{\text{Number of mergeable pairs}}{n-1} \tag{2}$$

This score measures the proportion of potentially unnecessary splits among all consecutive pairs. A lower fragmentation score indicates that the detected failure periods are more continuous and coherent, suggesting a better parameter setting for reflecting user-perceived failures.

This metric is evaluated across various combinations of moving average window sizes and detection thresholds. Table 2.12 summarizes the results for ChatGPT's user reports, including the original number of detected failure periods and the number after merging adjacent ones. As expected, parameter settings with smaller window sizes and lower thresholds (e.g., window size = 5, threshold = 0.6) tend to produce a large number of short segments, resulting in higher fragmentation. These settings are considered overly lenient, as they capture numerous minor fluctuations and noise. In contrast, larger windows and higher thresholds (e.g., window size = 9, threshold = 1.0) apply stricter criteria, thereby reducing both the number and length of extracted segments.

The results show that for all window sizes, the fragmentation score consistently reaches its minimum when the threshold is set to 0.8. Among these, both the (7, 0.8) and (9, 0.8) combinations achieve the lowest fragmentation score of 0.191, indicating that they best capture coherent and continuous failure periods without over-fragmentation. To

determine which of the two should be selected as the final setting, the trade-off between strictness and coverage is considered. While a larger window size of 9 imposes stricter criteria and results in fewer detected failure periods (205 original, 166 after merging), the slightly smaller window size of 7 yields more detected segments (273 original, 221 after merging) and thus increases the likelihood of capturing potentially overlapping periods with operator-reported failures. In this context, a more permissive setting (i.e., smaller window size) is preferable, as it emphasizes recall over precision, ensuring that possible failure events are not missed even if some borderline cases are included. Based on this reasoning, the combination of window size = 7 and threshold = 0.8 is selected as the optimal setting for extracting user-reported failure periods for ChatGPT.

Table 2.12: Sensitivity analysis results across different parameter combinations for moving average window size and detection threshold.

window_size	**threshold**	**fragmentation_score**	**original_periods**	**merged_periods**
5	0.600	0.284	532	381
5	0.700	0.251	268	201
5	**0.800**	**0.251**	268	201
5	0.900	0.291	176	125
5	1.000	0.291	176	125
6	0.600	0.273	557	405
6	0.700	0.211	324	256
6	**0.800**	**0.211**	324	256
6	0.900	0.269	228	167
6	1.000	0.269	228	167
7	0.600	0.221	412	321
7	0.700	0.221	412	321
7	**0.800**	**0.191**	273	221
7	0.900	0.232	204	157
7	1.000	0.232	204	157
8	0.600	0.269	502	367
8	0.700	0.219	324	253
8	**0.800**	**0.196**	236	190
8	0.900	0.201	185	148
8	1.000	0.201	185	148
9	0.600	0.195	371	299
9	0.700	0.199	278	223
9	**0.800**	**0.191**	205	166
9	0.900	0.206	176	140
9	1.000	0.206	176	140

Finally, all extracted time segments are stored in a new dataset with two columns: `start_time` and `end_time`, where each row represents a continuous failure period. For each period, the most frequently reported failure type is identified by examining all user reports that fall within the corresponding time window, based on the processed user report datasets (i.e., Table 2.9 and Table 2.11). This dominant failure type is then stored in a new column, `failure_type`. The resulting dataset of user-reported failure periods is illustrated in Table 2.13, which includes three columns: `start_time`, `end_time`, and `failure_type`.

Table 2.13: User-perceived failure periods with the most frequently reported failure types.

start_time	end_time	failure_type
2025-04-30 02:16:54	2025-04-30 02:57:17	Error Received
2025-04-30 03:14:20	2025-04-30 04:56:09	Login
2025-04-30 08:33:17	2025-04-30 09:03:50	Error Received
2025-04-30 14:48:04	2025-04-30 16:13:46	Slow
2025-04-30 16:54:39	2025-04-30 17:39:08	Error Received

2.3 Data Analysis

This section conducts a multi-faceted analysis of the processed failure data to evaluate the reliability of LLM services. It examines failure recovery patterns, temporal dynamics, inter-source correlations, and the consistency between operator- and user-reported failures. Each component addresses a distinct aspect of service failure behavior, ranging from recovery efficiency and temporal regularities to reporting alignment and discrepancies. Together, these analyses provide a quantitative foundation for comparing provider performance and understanding the reliability characteristics of LLM services.

2.3.1 Failure-Recovery Modeling

To characterize how failures manifest and evolve in LLM services, this section models key aspects such as recovery duration, failure interval, failure severity, and failure type. Drawing on failure data from two perspectives, the analysis applies quantitative metrics, including MTTR, MTBF, and others, to capture multiple dimensions of failure behavior. The goal is to establish a robust foundation for comparing failure recovery patterns across different LLM services and between the two reporting sources. This analysis addresses RQ2 and presents the corresponding observations in Section 3.

Impact of Failure Type and Severity on Recovery Time To analyze how different failure types and impact levels relate to service recovery time, this subsection presents two sets of analyses based on operator-reported and user-reported failure datasets (Table 2.8, Table 2.13). The recovery time of each failure is computed as the difference, in minutes, between `start_time` and `end_time`. Failures with a zero duration or lasting longer than 72 hours are excluded to mitigate data noise and outliers.

First, the relationship between failure types and recovery durations is examined by calculating the median recovery time per failure type for each LLM service, separately for operator and user reports. The resulting data are visualized as grouped bar charts (Figure 3.1), where each bar represents the median recovery time for a given failure type and LLM service.

To further capture the variability of recovery durations within each failure type, Figure 3.2 presents a set of boxplots comparing the distribution of recovery times across operator and user sources, broken down by LLM service and failure type. This visualization highlights both central tendencies and dispersion patterns, facilitating a comparative assessment of failure resolution behavior across services and reporting perspectives.

In addition to failure types, the relationship between failure impact level and recovery time is analyzed based solely on operator-reported incidents, as user reports do not contain impact information. Figure 3.3 displays boxen plots of recovery durations grouped by impact level and service, enabling the examination of how severity correlates with recovery efficiency. Boxen plots are particularly well-suited for this analysis, as recovery times tend to exhibit right-skewed, long-tailed distributions. This visualization provides a more detailed view of the upper quantiles while maintaining a clean appearance.

MTTR and MTBF Analysis To assess the recovery efficiency and failure frequency of LLM services, this subsection analyzes two widely used reliability metrics: Mean Time to Recovery (MTTR) and Mean Time Between Failures (MTBF). These metrics are computed separately using operator-reported and user-reported failure datasets (Table 2.8, Table 2.13). For each dataset, recovery time is calculated as the duration between the start and end timestamps of a failure event, while the time between failures is defined as the duration from the end of one failure to the start of the next. Failures with extremely long durations (greater than 10 days) are excluded to reduce noise.

Figure 3.4 presents the distributions of MTTR and MTBF across all LLM services using horizontal boxplots with logarithmic scales. Each box represents either operator-reported or user-reported statistics, grouped by service. This visualization highlights not only the central tendencies of the recovery and failure spacing patterns, but also the high degree of skewness present in both metrics. The use of a logarithmic axis enables consistent comparison across values ranging from minutes to days. Vertical reference lines are included to mark standard thresholds (e.g., 0.5 hours, 1 day) for easier interpretation.

To complement the boxplots, Figure 3.5 provides empirical cumulative distribution functions (ECDFs) of MTTR and MTBF, separately plotted for operator and user datasets. These ECDF plots enable fine-grained comparison of recovery and failure intervals across services by showing what proportion of failures or inter-failure gaps fall below a given threshold. Logarithmic axes are again applied to capture the wide range of observed values. This dual representation offers a robust and interpretable view of how quickly services recover and how frequently failures occur under different reporting perspectives.

Comparative Analysis of Failure Patterns To conduct a comparative analysis of the reliability and robustness of different LLM services, this work defines and computes a suite of metrics that reflect multiple dimensions of system behavior during and between service failures. These metrics are logically grouped to capture: (1) the duration and efficiency of recovery processes, (2) the temporal distribution and spacing between failures, (3) overall failure frequency and diversity, (4) system-wide availability, and (5) specific characteristics observable only from operator side, such as detection delay and the overall severity of failures. Each metric is defined as follows.

Mean Time to Recovery (MTTR). MTTR refers to the average amount of time required to restore a service after a failure has occurred. It measures the responsiveness and effectiveness of the recovery process. Given a set of N incident reports, each with a

start time $t_{\text{start}}^{(i)}$ and an end time $t_{\text{end}}^{(i)}$, the MTTR is computed as:

$$\text{MTTR} = \frac{1}{N} \sum_{i=1}^{N} \left(t_{\text{end}}^{(i)} - t_{\text{start}}^{(i)} \right) \tag{3}$$

Median Failure Duration (MFD). MFD captures the median duration of failure events, offering a robust central tendency measure that is less sensitive to extreme values. For each incident, the duration is computed as $(t_{\text{end}}^{(i)} - t_{\text{start}}^{(i)})$, and the MFD is the median of these values across all incidents.

Tail [P95] Failure Duration (TFD). TFD represents the 95th percentile of failure durations, characterizing the worst-case scenarios in terms of prolonged outages. It provides insight into the tail behavior of the system's recovery performance and is useful for risk assessment and SLA evaluation.

Mean Time Between Failures (Mean MTBF). MTBF indicates the average time interval between the end of one failure and the start of the next. It reflects the system's reliability or uptime. For a chronologically ordered sequence of N incidents, the MTBF is calculated as:

$$\text{MTBF} = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(t_{\text{start}}^{(i+1)} - t_{\text{end}}^{(i)} \right) \tag{4}$$

Median Time Between Failures (Median MTBF). This metric computes the median time between the end of one incident and the start of the next, based on all adjacent incident pairs. It provides a robust alternative to the mean MTBF that mitigates the influence of extremely long or short intervals.

Tail [P95] Time Between Failures (TTBF). TTBF quantifies the 95th percentile of time intervals between successive failures, offering insight into the longest stretches of continuous service availability.

Failure Frequency. Failure frequency refers to the number of failure incidents observed within the monitoring period. It serves as a basic but informative indicator of how often the system experiences observable disruptions.

Failure Type Entropy. This metric captures the diversity of failure types by computing the Shannon entropy over all observed failure categories. Let C be the set of distinct failure types and p_j the proportion of failures belonging to type j, the entropy is given by:

$$H = -\sum_{j \in C} p_j \log_2 p_j \tag{5}$$

As mentioned earlier, failure types are classified into six categories: *Login*, *Inaccessible*, *Slow*, *Error Received*, *Others*, and *Unknown*. A higher entropy value indicates a more heterogeneous distribution of failure types, suggesting a broader spectrum of issues.

Availability Percentage. Availability is computed as the fraction of total monitoring time during which the service is not affected by any reported failure. It is calculated as:

$$\text{Availability} = 1 - \frac{\sum_{i=1}^{N}(t_{\text{end}}^{(i)} - t_{\text{start}}^{(i)})}{T_{\text{total}}} \tag{6}$$

where T_{total} is the total duration of the observation period.

Average Failure Impact Level (Ordinal). This metric averages the ordinal impact level labels associated with each operator-reported failure. By assigning numeric values to impact levels (with higher numbers indicating greater severity), the overall severity of failures can be assessed over time. This metric is only available from operator reports.

2.3.2 Temporal Pattern Analysis

Understanding when failures are most likely to occur is essential for diagnosing the causes of service disruptions and designing effective mitigation strategies. This section explores temporal patterns in LLM service failures by analyzing their distribution over time. Specifically, it examines whether failures show periodic regularities, concentrate in specific time windows, or display temporal clustering. The analysis considers multiple time granularities, including daily, weekly, and hourly intervals, and applies techniques such as trend visualization, categorical time-based distributions, and autocorrelation analysis. By revealing patterns from both operator- and user-reported failures, this study investigates whether temporal signals can provide reliability insights to support proactive service management and enhance user experience. This analysis addresses RQ3 and presents the corresponding observations in Section 4.

Temporal Trends To analyze temporal trends in LLM service failures, this study computes daily and weekly failure counts using both operator-reported and user-reported failure datasets (Table 2.8, Table 2.13). Each failure event is counted based solely on its reported start time. For the daily trend, a failure contributes one count to the calendar day on which it began, regardless of its duration or whether it continued into subsequent days. Similarly, for the weekly trend, each failure is assigned to the calendar week containing its start time. This approach ensures that each failure is counted only once, using a consistent timestamp anchor across temporal resolutions. The resulting time series are aligned and plotted separately for operator and user reports, with one panel visualizing daily trends and another visualizing weekly trends. This design facilitates comparative assessment of failure frequency patterns across different temporal granularities and reporting sources.

Temporal Distributions To investigate whether service failures exhibit periodic patterns, this subsection analyzes their distributions across multiple time-based partitions—ranging from weekly cycles to intra-day variations—using both operator- and user-reported failure datasets (Table 2.8, Table 2.13). The occurrence time of each failure is defined as the calendar day or hour corresponding to its reported start time.

The first analysis compares failure frequencies between weekdays and weekends. Failures are grouped by binary weekday category, allowing for an assessment of whether service instability is more prevalent during workdays, which may experience higher usage demands.

To ensure comparability given the unequal number of weekdays (five) and weekends (two), the comparison is based on the average number of failures per day within each category.

The second analysis examines failure distributions across individual weekdays (Monday through Sunday), with the goal of identifying systematic patterns over the seven-day weekly cycle—for example, whether certain days such as Fridays are more failure-prone.

The final analysis explores hourly variation by grouping failures according to the hour of day in which they began. This component is motivated by user concerns about degraded service quality during specific times of day—particularly morning hours in high-demand regions such as the U.S. West Coast. Aggregating failures by hour enables the identification of diurnal patterns in outage occurrence across different services.

Auto-correlations To examine whether service failures exhibit temporal self-dependency, this subsection applies autocorrelation function (ACF) analysis to the time series of failure occurrences. ACF measures the correlation of a time series with lagged versions of itself, offering insight into whether the presence of a failure on a given day or week increases the likelihood of another failure occurring at regular intervals. This can reveal latent periodicities or persistence effects in service instability patterns.

For each LLM service, failure time series are constructed by aggregating the number of failures per day and per week, using the start time of each failure as the temporal anchor. Separate analyses are conducted for operator-reported and user-reported data (Table 2.8, Table 2.13). For daily autocorrelations, the time series is resampled at a one-day interval; for weekly autocorrelations, it is resampled at a one-week interval. Missing dates or weeks without reported failures are assigned zero counts to maintain a continuous timeline.

The number of lags used in the ACF plots is determined by the available time range in each dataset. For operator-reported data, up to 80 daily lags and 50 weekly lags are applied. In contrast, user-reported data spans only a three-month period and therefore uses fewer lags—40 for daily and 8 for weekly autocorrelations. ACF plots are then generated for each service and reporting perspective to visualize autocorrelation structures over both short-term and long-term intervals.

2.3.3 Correlation Analysis

This section analyzes two types of correlation between operator and user perspectives. The first examines the statistical relationship between user-reported issue volume and the presence of official incidents, using correlation coefficients to quantify their association. The second investigates failure co-occurrence to evaluate whether failures tend to align across LLM services. Such exploration offer insights into the level of agreement between reporting sources and the potential infrastructure interdependencies among services. This analysis addresses RQ4 and presents the corresponding observations in Section 5.

Correlation Between User Reports and Official Incidents To quantify the relationship between user-reported issue volume and officially disclosed incidents, both Pearson and Spearman correlation coefficients were computed separately for each LLM service. The time series were constructed at a 5-minute resolution. Within each window, the number of user reports was aggregated based on the user report count dataset (Table 2.10), and a binary indicator was assigned to denote whether an official incident was active during

that interval, using the operator-reported failure datasets (Table 2.8). Pearson correlation was used to measure linear dependence between the two signals, while Spearman correlation assessed monotonic (rank-based) associations. Statistical significance was annotated using standard asterisk notation: $^{*}p < 0.05$, $^{**}p < 0.01$, and $^{***}p < 0.001$.

Co-occurrence of Failures To investigate whether service outages tend to occur simultaneously across different LLM services, a co-occurrence analysis was conducted based on daily failure events. The analysis was performed separately for the operator-reported and user-reported failure datasets (Table 2.8, Table 2.13), both of which provide timestamped records of service disruptions.

For each service, the set of failure dates was extracted by using only the reported `start_time` of each incident to determine the day of failure. A day was considered a failure day for a given service if any failure began on that date.

Let S_i and S_j denote two services. A co-occurrence count matrix C was computed, where each element C_{ij} represents the number of calendar days on which both S_i and S_j experienced a failure. This symmetric matrix captures the raw overlap in failure days between service pairs.

To further analyze directional dependencies, a conditional probability matrix P was calculated, where each element P_{ij} denotes the probability that service S_i experiences a failure on a day, given that service S_j fails on that day. This is formally defined as:

$$P_{ij} = P(S_i \mid S_j) = \frac{|\mathcal{D}_{S_i} \cap \mathcal{D}_{S_j}|}{|\mathcal{D}_{S_j}|} \tag{7}$$

where $\mathcal{D}_{S_k}$ denotes the set of failure days for service S_k.

In the resulting heatmap visualizations, the rows (y-axis) represent the conditional target service S_i, the columns (x-axis) represent the conditioning service S_j, and the cells display $P(S_i \mid S_j)$ as percentages. This formulation provides a comparative view of inter-service failure correlations and potential operational dependencies.

2.3.4 Consistency Analysis

Because consistency between operator disclosures and user experiences is essential for comprehensively evaluating LLM service reliability, this section investigates the alignment between operator- and user-reported failures across two dimensions. First, it examines whether the failure types acknowledged by service providers correspond to those perceived and reported by users. Second, it evaluates the extent to which the failure periods reported by LLM providers match user-reported timelines. The results of these comparisons reveal varying levels of consistency across providers, which may reflect differences in monitoring practices, disclosure policies, or responsiveness to user feedback. This analysis addresses RQ5 and presents the corresponding observations in Section 6.

Consistency of Failure Types To evaluate the consistency of failure type classifications between operator and user reports, this subsection introduces two complementary analyses, both using operator-reported failure datasets (Table 2.8) and raw user report datasets (Table 2.5). The goal is to assess the extent to which user-perceived failure types align with the categories reported by LLM service providers.

The first analysis calculates the match rate for each service. For every operator-reported failure window, the most frequently reported failure type among all temporally overlapping user reports is identified. If this dominant user-reported type matches the operator-assigned failure type, it is counted as a match. The match rate is then computed as the percentage of such matches relative to the total number of operator-reported failures. This metric offers a high-level summary of the agreement between user and operator perspectives across services.

The second analysis provides a more detailed view of the mapping between operator and user classifications. For each service, a confusion matrix is constructed using operator-reported failure windows with corresponding user-reported issues. Within each window, the most frequently reported user failure type is paired with the operator-reported failure type, and the frequency of each type-to-type pairing is recorded. The resulting heatmaps reveal common agreement patterns and recurrent misalignments between the two sources. Collectively, these analyses enable an assessment of classification consistency and potential gaps in mutual understanding of failure types.

Consistency of Failure Periods To evaluate the alignment between operator- and user-reported failure periods, this subsection compares the temporal overlap of failure windows across the two sources. For each LLM service, a side-by-side timeline visualization is constructed, displaying operator-reported and user-reported failures along a shared calendar axis. Operator-reported failure windows are color-coded by impact level, while user-reported failure windows are rendered as parallel bars in a uniform color. This layout facilitates visual inspection of overlaps, discrepancies, and potential reporting lags between the two perspectives.

In addition to visual inspection, the analysis incorporates quantitative comparisons using consistency-related metrics, such as the percentage of overlapping reports and the lead or lag time of user-reported failures relative to corresponding operator reports. By systematically comparing operator and user perspectives, this study aims to uncover reporting discrepancies and identify providers whose official disclosures closely reflect user-reported issues. It also highlights services where user feedback deviates significantly from operator acknowledgments. These findings offer insight into the consistency and responsiveness of provider-side reporting in relation to actual user experience.

3 Failure-Recovery Modeling

This section models and compares the failure recovery characteristics of major LLM services using both operator and user reports, aiming to reveal patterns in failure behavior across services and between data sources. The analysis includes: (1) how recovery time varies across failure types and impact levels; (2) analysis and comparison of MTTR and MTBF distributions across services; and (3) comparative evaluation of failure metrics across all services and perspectives.

3.1 Impact of Failure Type on Recovery Time

Observation #1: *In operator reports, Slow failures generally show the longest median recovery durations. Averaged across services, the median recovery time for this failure type is 115.5 minutes, compared to 64.8 for Login, 60.8 for Inaccessible, and 67.3 for Error Received failures.*
Observation #2: *Operator reports exhibit greater variability in median recovery durations, both within and across failure types (average within-type variance: 1585 minutes^2; across-type: 1839 minutes^2). In contrast, user reports show much lower variability (631 minutes^2 within-type; 377 minutes^2 across-type), indicating more consistent user perceptions of recovery durations.*

Figure 3.1 presents the median recovery durations for four major failure types across operator and user reports. In operator data, *Slow* failures generally incur the longest recovery times (Observation 1). For example, Character.AI reports a median duration of 183 minutes for *Slow* failures—2.5x as long as its *Inaccessible* and *Error Received* failures (both 72 minutes) and nearly 2x that of *Login* (94 minutes). The only exception to this trend is observed in ChatGPT, where *Inaccessible* failures reach 139 minutes, nearly 2x its *Slow* duration of 71 minutes. This pattern suggests that *Slow* failures may involve more complex performance degradations or backend bottlenecks that are harder to isolate and resolve than access-related issues such as *Login* or *Inaccessible* failures.

Operator-reported medians also demonstrate greater variability both within and across services (Observation 2). For instance, DeepSeek's operator reports range from just 15 minutes for *Inaccessible* failures to 105 minutes for *Slow*—a 7x difference. In contrast, user-reported recovery durations exhibit a relatively narrow range across both failure types and services, with most values falling between 50 and 70 minutes. An exception is DeepSeek's *Login* failures, where the user-reported median reaches 135 minutes—1.9x its *Slow* duration (70 minutes). This divergence between operator and user data likely reflects differences in reporting granularity and visibility. Operators track recovery from the system's perspective, often with fine-grained logs tied to specific components. Users, however, may not perceive the recovery until full functionality is restored on the interface level. As a result, users tend to report recovery durations in a more compressed and homogeneous range, even when underlying failure types vary substantially.

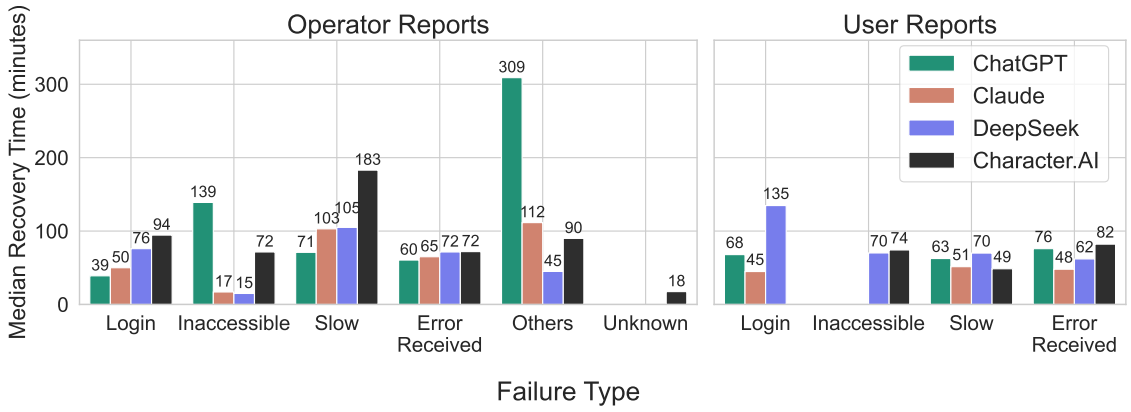


Figure 3.1: Median recovery time by failure type across LLM services. The left plot shows results based on operator reports, while the right plot shows results based on user reports.

Observation #3: *Slow and Error Received failures tend to show higher dispersion in recovery time distributions, characterized by broader interquartile ranges, with average IQRs of 77 and 93 minutes, respectively, and more frequent outliers across both sources and services.*

Observation #4: *Claude exhibits larger discrepancies between operator and user recovery time distributions across failure types (average IQR: 117 vs. 29 minutes), whereas ChatGPT displays more consistent patterns between the two sources (111 vs. 112 minutes, respectively).*

Figure 3.2 further reveals the dispersion patterns underlying these medians. *Slow* and *Error Received* failures generally exhibit broader interquartile ranges and more frequent outliers compared to *Login* and *Inaccessible* (Observation 3). For example, Claude's operator reports show wide spreads for both types, with upper whiskers extending beyond 300 minutes and multiple outliers—unlike the much tighter and more contained distributions observed for *Login* and *Inaccessible* failures. This observation suggests that *Slow* and *Error Received* failures may correspond to more operationally heterogeneous incident classes. Depending on root causes, such failures can either be addressed quickly (e.g., through cache clearing or request rerouting) or escalate into protracted service degradations. In contrast, *Login* and *Inaccessible* failures may stem from more easily identifiable and resolvable subsystems, leading to more consistent recovery durations.

The comparison between operator and user distributions also highlights differences in failure reporting consistency across services (Observation 4). Claude exhibits the most pronounced divergence: operator distributions are wide and skewed across most failure types, while user reports remain tightly clustered—for instance, Claude's operator-reported *Slow* failures have a long right tail, whereas user reports center closely around 51 minutes. In contrast, ChatGPT shows much stronger alignment between sources, as the operator and user distributions display similar box lengths, whisker ranges, and frequencies of outliers. These discrepancies may reflect differences in user reporting practices. One possible explanation is that Claude's relatively smaller user base leads to sparser reporting during incidents, with user submissions only capturing part of the full recovery window. As a result, user-side recovery durations appear shorter and more concentrated than those in

operator reports. By comparison, ChatGPT's internal and external reporting appear more aligned, likely because its larger user population generates denser feedback that more fully captures the recovery timeline.

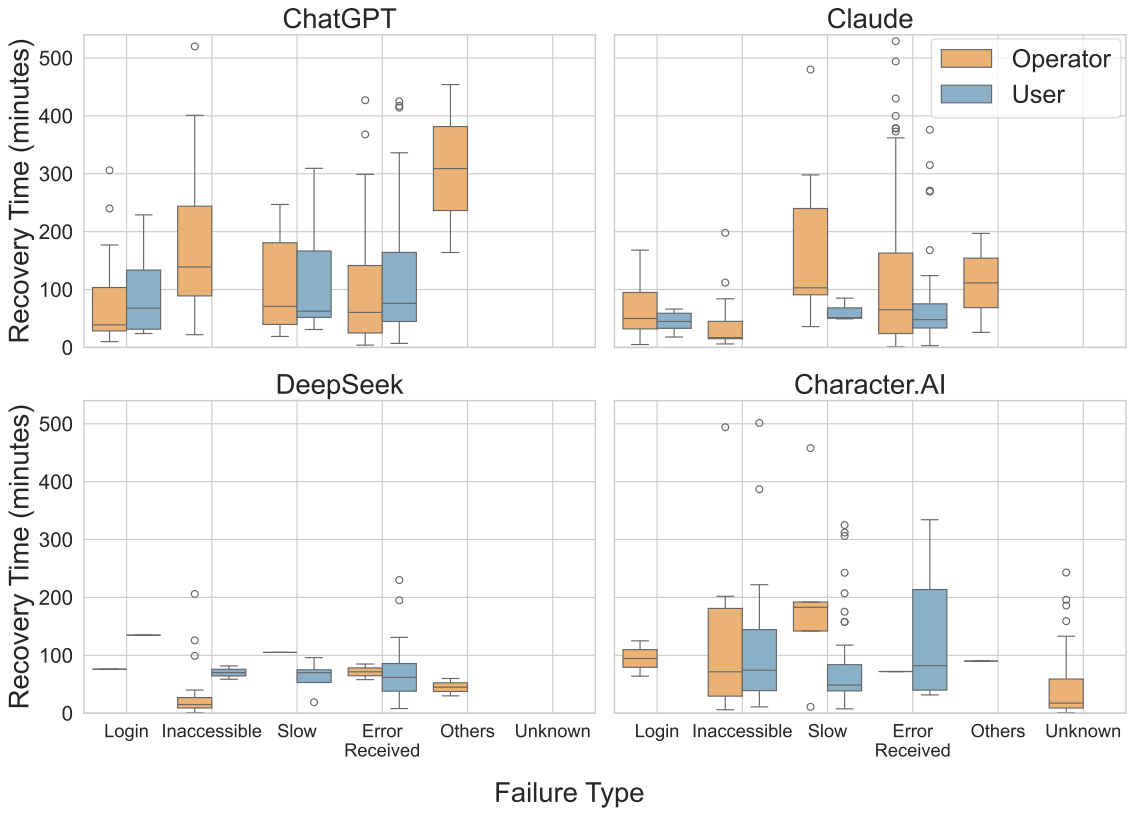


Figure 3.2: Recovery time distributions by failure type across LLM services, comparing operator- and user-reported failures to highlight differences in recovery patterns.

3.2 Impact of Failure Severity on Recovery Time

Observation #5: *Across all services, failures labeled as Degraded Performance tend to exhibit the most dispersed recovery time distributions, as evidenced by up to five boxen layers, with the top edge reaching the 93.75th percentile, and frequent outliers. In contrast, Full Outage failures are more concentrated around the median, showing only the main interquartile box (25%–75%), which is likely a result of their lower distributional spread.*

Figure 3.3 shows recovery time distributions categorized by failure impact level across the four LLM services, based on operator reports. As illustrated, recovery times for failures labeled as *Degraded Performance* are not only longer on average but also substantially more variable—especially for ChatGPT and Claude. For instance, Claude's *Degraded Performance* incidents display a notably wide spread, with the uppermost box segment near the 93.75th percentile approaching 500 minutes and a dense concentration of outliers. ChatGPT exhibits a similar pattern, though with a slightly narrower range.

In contrast, *Full Outage* incidents—despite being ostensibly more severe—tend to exhibit shorter and more tightly distributed recovery durations. This is particularly evident in Claude and DeepSeek, where *Full Outage* recovery durations are often centered under 100 minutes. One possible explanation is that full outages trigger faster, more coordinated response efforts due to their visibility and broader impact, whereas degraded states may take longer to diagnose and resolve due to their ambiguous symptoms or localized scope.

Across services, this pattern is consistent: *Degraded Performance* incidents show higher variance and longer tails, while *Full Outage* incidents appear more contained (Observation 5). This suggests that failures labeled as degraded may span a wide spectrum of underlying issues—some transient, others prolonged. Such incidents are often characterized by ambiguous or intermittent symptoms, localized degradation, or unclear system boundaries, making it harder for operators to detect, diagnose, and prioritize them promptly. Meanwhile, full outages typically involve system-wide service disruption, where the impact is immediate, clearly visible, and broadly recognized. This clarity often enables faster detection, standard triage procedures, and more decisive mitigation workflows, resulting in shorter and more consistent recovery durations.

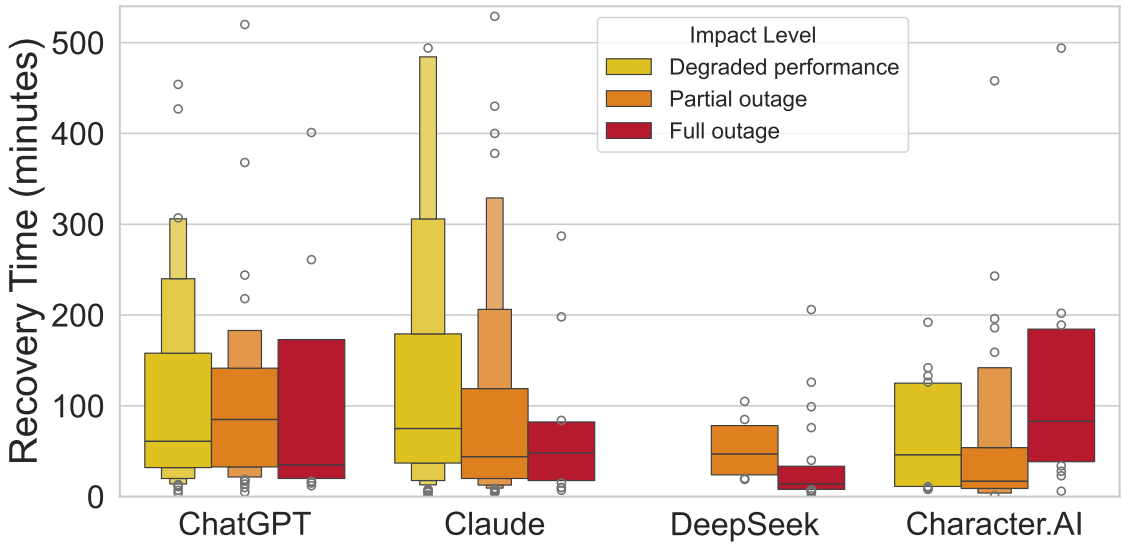


Figure 3.3: Recovery time distributions by failure impact level across LLM services, based on operator reports.

3.3 MTTR and MTBF Analysis

Observation #6: *ChatGPT recovers the slowest from failures, with the highest median MTTR reported by both the operator (1.07 hours) and the user (1.23 hours).*
Observation #7: *DeepSeek exhibits the largest discrepancy in median MTTR between operator and user reports (0.81-hour difference), whereas ChatGPT demonstrates the highest alignment (0.16-hour difference).*
Observation #8: *DeepSeek is the most reliable service on the operator side, with the lowest median MTTR (0.40 hours) and the highest median MTBF (5.39 days), whereas Claude ranks highest in user-side reliability (median MTTR: 0.82 hours; median MTBF: 1.07 days).*
Observation #9: *DeepSeek also exhibits the largest divergence in median MTBF between operator and user reports (4.6-day difference), whereas Claude shows the highest consistency (0.11-day difference).*

Figure 3.4a presents the distribution of MTTR across services and data sources. Among the four services, ChatGPT stands out with the longest recovery durations overall: both operator- and user-reported median MTTRs are the highest (1.07h and 1.23h, respectively), suggesting consistently slow recovery performance (Observation 6). However, from an alignment perspective, ChatGPT displays the closest match (0.16h difference) in median MTTRs between perspectives (Observation 7).

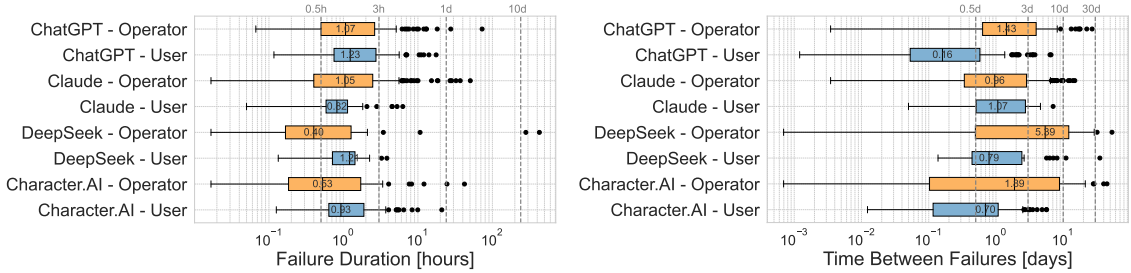
At the other end of the spectrum, DeepSeek shows the shortest operator-reported MTTR (0.40h) but a nearly longest user-reported MTTR (1.21h), revealing a notable misalignment between official reports and user perception (Observation 7). This 0.81-hour gap—the widest among all services—suggests that user-facing availability does not always reflect backend service recovery timelines.

Figure 3.4b shows the distribution of MTBF across services. From the operator perspective, DeepSeek presents a much higher median MTBF (5.39d) than other services, along with the lowest median MTTR (0.40h), indicating infrequent and quickly resolved failures. This suggests that it is the most reliable service (Observation 8). However, this operator-reported reliability is not fully reflected in user reports. Median MTBF of DeepSeek drops sharply to 0.79 days from the user perspective, marking a 4.6-day difference between the two sources—the largest discrepancy across all services (Observation 9). This gap may be explained by DeepSeek's open-source nature: users often encounter "server busy" messages during periods of high traffic and report such instances as failures, even though these events are typically not logged as incidents by the operator. As a result, user-perceived disruptions occur more frequently than officially recorded failures, leading to a significant divergence in median MTBFs.

Looking at the user side, Claude ranks highest in median MTBF (1.07d), and also shows the lowest median MTTR (0.82h), suggesting that users experience fewer and shorter failures. Together, these indicate that Claude has the best user-side reliability (Observation 8). Additionally, Claude exhibits the highest agreement in median MTBF between operator and user reports (0.96d and 1.07d, respectively). The small 0.11-day difference implies consistency in failure intervals between internal system records and external user experience (Observation 9).

ChatGPT, while moderately stable from the operator side (1.43d), shows a much shorter

median MTBF from the user perspective (0.16d), which may be due to its larger user base, increasing the likelihood that minor disruptions are noticed and reported.



(a) MTTR distributions across LLM services.

(b) MTBF distributions across LLM services.

Figure 3.4: Distributions of mean time to recovery (MTTR) and mean time between failures (MTBF) across four major LLM services, based on operator and user reports. The horizontal axis is logarithmic.

Observation #10: *On the operator side, DeepSeek resolves failures most quickly (60.61% of durations $\leq$ 0.5 hours) and experiences the longest intervals between them (47.22% of intervals $>$ 7 days), as reflected by its steep MTTR ECDF and right-shifted MTBF ECDF. In contrast, ChatGPT and Claude recover more slowly and exhibit more frequent failures.*
Observation #11: *From the user perspective, Claude exhibits the fastest recovery (70.59% of durations $\leq$ 1 hour) and the longest failure intervals (52.00% of intervals $>$ 1 day). ChatGPT again shows slower recovery and more frequent user-reported failures.*

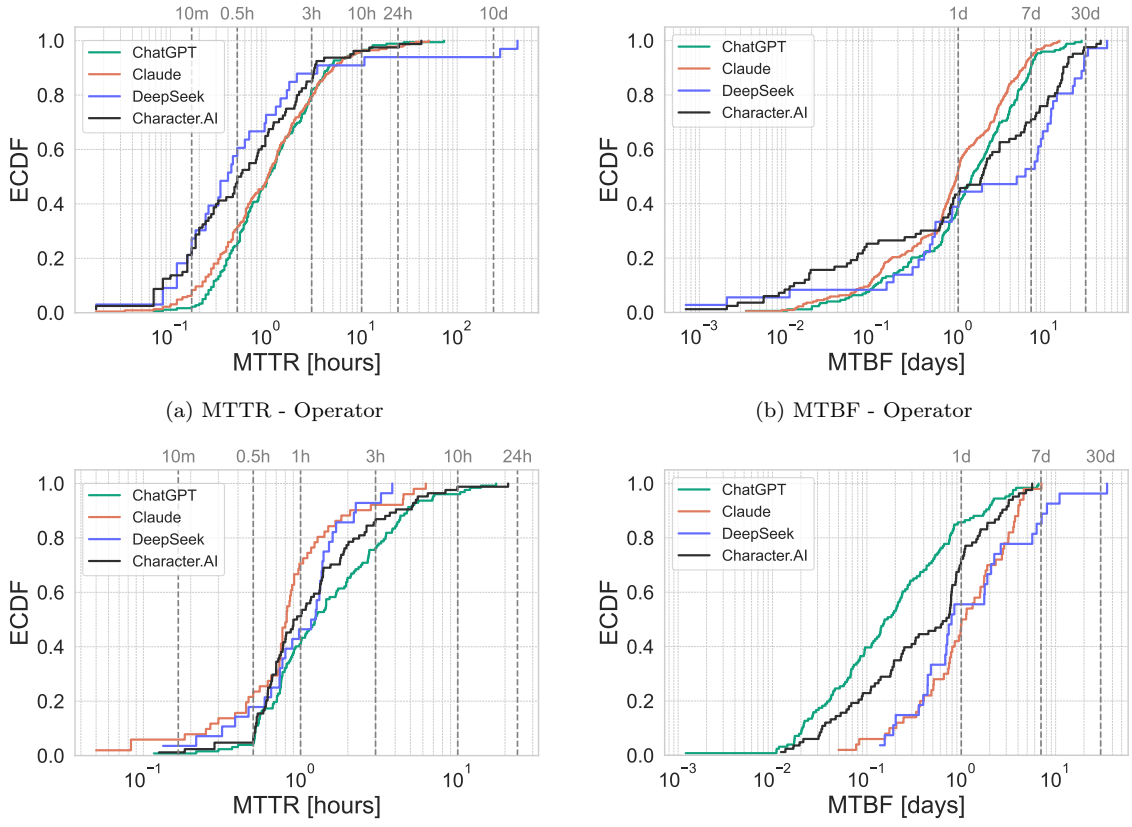
Figure 3.5 presents empirical cumulative distribution functions (ECDFs) for MTTR and MTBF across the four services, offering a fine-grained view of failure recovery and recurrence patterns. From the operator perspective, DeepSeek exhibits the fastest recovery performance. As shown in Fig. 3.5a, 60.61% of its failures are resolved within just 0.5 hours, and 87.88% within 3 hours (Observation 10). Claude and ChatGPT recover significantly more slowly: at the 0.5-hour mark, only 31.72% of Claude's incidents and 25.70% of ChatGPT's are resolved (Observation 10). However, the gap narrows considerably at the 3-hour mark—ChatGPT reaches 82.12%, while Claude lags slightly behind at 79.30%. Character.AI falls between these extremes, with 50.00% of cases resolved in 0.5 hours and 85.00% within 3 hours.

The operator-side MTBF distributions (Fig. 3.5b) further differentiate reliability patterns. DeepSeek again stands out: only 38.89% of its operator-reported failures occur within 1 day, and 47.22% occur after 7 days—suggesting high service stability (Observation 10). In contrast, Claude and ChatGPT show much steeper ECDFs, indicating more frequent failures: 52.49% of Claude's failures and 38.73% of ChatGPT's occur within just 1 day, rising to 94.57% and 90.17%, respectively, within 7 days (Observation 10). Character.AI exhibits a more moderate pattern, with 43.37% of failures occurring within 1 day and 71.08% within 7 days. These trends reinforce DeepSeek's robust service reliability, while highlighting the relatively frequent failures encountered by Claude and ChatGPT.

In user-perceived recovery (Fig. 3.5c), Claude exhibits the steepest ECDF curve, with 70.59% of user-reported failures resolved within 1 hour and 92.16% within 3 hours—demonstrating the fastest user-side recovery across all services (Observation 11). DeepSeek

and Character.AI follow, with 46.43% and 51.19% of failures resolved within 1 hour, and 92.86% and 86.90% resolved within 3 hours, respectively. ChatGPT again trails behind: only 76.38% of its failures are resolved within 3 hours, and the curve rises more gradually over time (Observation 11).

The user-side MTBF distributions (Fig.3.5d) reveal similar contrasts. Claude's curve remains the steepest, with only 48.00% of failure intervals within 1 day (Observation11), but it rises rapidly to 98.00% within 7 days—suggesting that 50% of the intervals are tightly concentrated within a 1–7 day range. In contrast, ChatGPT shows an earlier rise: 85.71% of failure intervals are under 1 day, and all fall within 7 days, indicating that ChatGPT experiences failures at a higher frequency (Observation 11). DeepSeek again displays the widest spread, with 55.56% of failure intervals under 1 day and 85.19% within 7 days. Character.AI lies between, with 72.29% of intervals in 1 day and all within 7 days. These results indicate that Claude has the most consistent failure pattern from the user's perspective, with most failures recurring within a narrow time frame. In contrast, ChatGPT and DeepSeek show different distributions: ChatGPT's failures tend to recur more quickly, while DeepSeek's failures are more spread out over time.



(a) MTTR - Operator

(b) MTBF - Operator

(c) MTTR - User

(d) MTBF - User

Figure 3.5: ECDFs of MTTR and MTBF across LLM services, based on operator and user reports. A curve closer to the upper left indicates shorter durations, suggesting faster recovery (for MTTR) or more frequent failures (for MTBF).

3.4 Comparative Analysis of Failure Patterns

Observation #12: *DeepSeek exhibits a large discrepancy in the mean–median MTTR gap between operator and user reports (21.984-hour vs. 0.022-hour difference), suggesting that a few extremely prolonged failures were reported by the operator but perceived as intermittent disruptions by users.*
Observation #13: *ChatGPT shows the largest entropy gap between operator and user reports (0.702-bit difference), indicating that users tend to report failures of certain types.*
Observation #14: *Character.AI's low failure frequency (0.165/day) in operator reports coexists with a relatively high average failure impact level (1.942), suggesting that failures are infrequent but severe when they do occur.*
Observation #15: *In terms of failure frequency and service availability, Character.AI is the most reliable service from the operator perspective (0.165/day, 98.699%), whereas DeepSeek ranks highest on the user side (0.293/day, 98.497%).*

Table 3.1 and Table 3.2 integrate multiple failure metrics, including recovery duration, failure frequency, availability, impact severity, and failure type diversity, across both operator and user perspectives. The analysis reveals marked differences not only across services but also between how reliability is measured internally versus perceived externally.

A pronounced divergence is observed in DeepSeek's reported recovery durations. The operator-side mean MTTR is 22.384 hours, while the median MTTR is only 0.4 hours, yielding a huge gap of 21.984 hours, the largest among all services. In contrast, user reports show a mean MTTR of 1.230 hours and a median of 1.208 hours, resulting in a minimal gap of 0.022 hours. This suggests that a few exceptionally long incidents reported by the provider were not perceived by users as continuous disruptions (Observation 12). These incidents likely manifested as intermittent symptoms—such as repeated server busyness or region-specific unavailability—that fragmented the user experience. By comparison, Claude's operator-reported MTTR shows a more balanced distribution (mean: 2.600h; median: 1.050h), and ChatGPT likewise maintains a modest gap (2.504h vs. 1.067h), reinforcing that DeepSeek's asymmetry is uniquely severe.

Differences in failure type diversity further illustrate perception gaps. Among all services, ChatGPT exhibits the largest entropy difference between operator and user reports—1.444 bits vs. 0.742 bits, a gap of 0.702 bits (Observation 13). This reflects a user tendency to report various encountered issues under specific failure types—particularly *Error Received*—whereas the provider tends to describe failures more precisely. Other services display smaller entropy gaps: Claude (1.029 vs. 0.978 = 0.051 bits), DeepSeek (1.165 vs. 1.234 = 0.069 bits), and Character.AI (1.400 vs. 1.234 = 0.166 bits).

Another revealing asymmetry appears in Character.AI's failure pattern. It shows the highest operator-reported availability at 98.699% and a relatively low failure frequency of 0.165 failures/day. However, the average failure impact level (ordinal) is relatively high at 1.942 (Observation 14). This pattern suggests that although Character.AI rarely experiences failures, when disruptions do occur, they tend to be high-impact events, possibly affecting major components or leading to full outages. In contrast, ChatGPT's average impact level is the lowest (1.458), and Claude reports a moderate 1.584. These results point to contrasting failure patterns: Character.AI maintains high uptime but may be more vulnerable to severe disruptions, whereas ChatGPT experiences failures more frequently,

though with less severe impact.

A clear distinction emerges when comparing failure frequency and availability across services. From the operator side, DeepSeek reports the lowest failure frequency (0.099/day), but surprisingly also shows the lowest availability (91.763%). As mentioned earlier, this suggests the presence of extremely large outliers in operator-reported downtime. Character.AI, by contrast, combines relatively low failure frequency (0.165/day) with the highest availability across all services (98.699%), reflecting rare and short-lived incidents (Observation 15). ChatGPT (0.338/day, 96.472%) and Claude (0.471/day, 94.899%) fall in the middle, with Claude showing the most frequent failures among services, but still maintaining reasonably high availability, likely due to faster recovery. From the user perspective, the rankings shift. DeepSeek stands out in user-side reliability, with the lowest failure frequency (0.293/day) and the highest availability (98.497%) (Observation 15). However, Character.AI shows a notable decline in user-perceived reliability—its failure frequency increases 5.8× to 0.951/day, and availability drops to 92.597%, suggesting that many user-facing issues go unreflected in operator reports. ChatGPT performs worst on the user side, with a striking 1.426 failures per day and availability falling to 86.421%, indicating that ChatGPT users frequently experience and report issues.

Table 3.1: Comparative failure metrics of LLM services based on operator reports. Legend: h = hour(s), d = day(s), bits = entropy units measuring failure type diversity; higher values indicate greater variability. The best-performing service for each metric is bolded.

Metric	**ChatGPT**	**Claude**	**DeepSeek**	**Character.AI**
Mean Time to Recovery (MTTR) [h]	2.504	2.600	22.384	**2.056**
Median Failure Duration (MFD) [h]	1.067	1.050	**0.400**	0.525
Tail [P95] Failure Duration (TFD) [h]	**7.677**	8.163	118.927	**7.663**
Mean Time Between Failures (Mean MTBF) [d]	2.951	2.069	**9.370**	6.076
Median Time Between Failures (Median MTBF) [d]	1.431	0.960	**5.393**	1.885
Tail [P95] Time Between Failures (TTBF) [d]	8.039	7.413	**29.666**	21.064
Failure Frequency [failures/day]	0.338	0.471	**0.099**	0.165
Failure Type Entropy [bits]	**1.444**	1.029	1.165	1.400
Availability Percentage [%]	96.472	94.899	91.763	**98.699**
Avg. Failure Impact Level (Ordinal)	**1.458**	1.584	2.600	1.942

Table 3.2: Comparative failure metrics of LLM services based on user reports.

Metric	**ChatGPT**	**Claude**	**DeepSeek**	**Character.AI**
Mean Time to Recovery (MTTR) [h]	2.285	**1.179**	1.230	1.869
Median Failure Duration (MFD) [h]	1.233	**0.817**	1.208	0.925
Tail [P95] Failure Duration (TFD) [h]	6.938	4.500	**2.899**	5.548
Mean Time Between Failures (Mean MTBF) [d]	0.584	1.697	**3.438**	0.981
Median Time Between Failures (Median MTBF) [d]	0.160	**1.066**	0.789	0.698
Tail [P95] Time Between Failures (TTBF) [d]	2.757	4.396	**10.135**	3.491
Failure Frequency [failures/day]	1.426	0.552	**0.293**	0.951
Failure Type Entropy [bits]	0.742	0.978	**1.234**	**1.234**
Availability Percentage [%]	86.421	97.291	**98.497**	92.597

3.5 Summary of Failure-Recovery Modeling

This section addresses RQ2 by modeling failure recovery behaviors across LLM services using both operator and user reports. The analysis explores how failure types, impact levels, and reporting perspectives shape recovery durations, and highlights service-specific patterns and cross-source differences revealed through multiple failure metrics.

3.5.1 Key Observations

In operator reports, failures of type *Slow* have the longest median recovery duration (115.5 minutes), followed by *Login* (64.8), *Inaccessible* (60.8), and *Error Received* (67.3) (Observation 1). Variability in median recovery durations is higher in operator reports, both within and across failure types (average within-type variance: 1585 minutes^2; across-type: 1839), compared to user reports (631 and 377 respectively) (Observation 2). *Slow* and *Error Received* failures show high dispersion in recovery time distributions, with average IQRs of 77 and 93 minutes (Observation 3). Claude has the largest operator–user IQR gap across failure types (average IQR: 117 vs. 29 minutes), while ChatGPT is more aligned (111 vs. 112) (Observation 4). In terms of failure impact level, recovery time distributions are most dispersed for *Degraded Performance* failures (Observation 5).

ChatGPT has the highest median MTTR from both perspectives (1.07 hours operator, 1.23 hours user) (Observation 6). DeepSeek shows the largest median MTTR gap between sources (0.81 hours), while ChatGPT shows the smallest (0.16 hours) (Observation 7). DeepSeek also exhibits the largest median MTBF gap (4.6 days), whereas Claude is the most consistent (0.11 days) (Observation 9). Despite these cross-source discrepancies, DeepSeek ranks highest in operator-side reliability (median MTTR: 0.40 hours; median MTBF: 5.39 days), whereas Claude performs best in user reports (0.82 hours and 1.07 days, respectively) (Observation 8).

ECDFs further support these trends: 60.61% of DeepSeek's operator-reported failures are resolved within 0.5 hours, and 47.22% of its failure intervals exceed 7 days (Observation 10); Claude leads in user-side recovery, with 70.59% of failures resolved within 1 hour and 50.00% of intervals falling within 1–7 days (Observation 11).

Finally, based on multiple failure metrics, DeepSeek exhibits a substantial discrepancy in the mean–median MTTR gap between operator and user reports (21.984 vs. 0.022 hours), suggesting mismatched perceptions of prolonged operator failures (Observation 12). ChatGPT has the largest failure type entropy gap between sources (0.702 bits) (Observation 13). Character.AI shows low failure frequency on the operator side (0.165/day) but high average failure impact (1.942) (Observation 14). In terms of availability, Character.AI ranks highest from the operator perspective (98.699%), whereas DeepSeek leads from the user side (98.497%) (Observation 15).

3.5.2 Use Cases Derived from Observations

Prioritized Incident Triage. Based on Observation 1, different failure types exhibit varying recovery durations, which can guide triage prioritization during incident response. For example, when a *Slow* failure is detected, which typically has the longest median recovery time, teams may initiate escalation procedures such as notifying senior engineers

or postponing non-critical deployments. This targeted response can help mitigate the risk of prolonged service degradation and support faster recovery.

Enhancing Duration Estimation with Stable User Signals. Based on Observation 2, the lower variability in user-reported recovery durations across and within failure types suggests that these signals can serve as a stable baseline for estimating incident durations. Their consistency also enables providers to cross-validate internal estimates and detect potential outliers in operator reports, supporting more robust diagnostics and post-incident analyses.

SLA-Aware Runtime Decision-Making. Based on Observations 6, 8, 10, and 11, differences in recovery durations and failure intervals across LLM services can guide SLA-aware runtime decisions based on service-specific reliability characteristics. For example, applications that require rapid recovery may select Claude as the preferred backend during time-sensitive operations, whereas systems relying on ChatGPT might introduce additional redundancy or buffering due to its slower recovery. In contrast, DeepSeek's strong reliability from the operator perspective may allow for more aggressive runtime optimizations without compromising availability.

Monitoring Cross-Perspective Discrepancies as Diagnostic Signals. Based on Observations 7 and 9, large differences in median MTTR or MTBF between operator and user reports, such as those observed in DeepSeek, reflect inconsistencies in perceived reliability between the two perspectives. While such discrepancies do not indicate which side is more accurate, they serve as useful diagnostic indicators for evaluating the completeness and accuracy of internal failure detection and duration estimation. By defining thresholds for acceptable cross-perspective gaps, providers can trigger internal audits when discrepancies persist or exceed predefined limits, thereby improving visibility into potential blind spots or misalignments in failure reporting practices.

Outlier-Aware Incident Attribution. Based on Observation 12, large gaps between the mean and median MTTR in operator reports, as seen in DeepSeek, suggest the presence of rare but extremely prolonged failures. When users do not perceive these incidents as continuous disruptions, it indicates fragmentation in how the failures are experienced. To improve attribution accuracy, reliability teams can adopt outlier-aware attribution logic that avoids treating all prolonged failures as continuous outages by default. Instead, this logic incorporates user-perceived impact to distinguish between genuinely persistent failures and intermittent disruptions. Such differentiation enables more accurate SLA evaluations and more effective root cause analysis.

Monitoring Entropy Gaps for Perception Shifts. Based on Observation 13, the high entropy gap between operator and user failure type distributions, as seen in ChatGPT, reflects a divergence between cause-based classification by providers and symptom-based reporting by users. Rather than attempting to reconcile the two sources, providers can monitor entropy gaps as a signal for shifts in user perception. For example, a sudden

increase in entropy gap may indicate that users are disproportionately reporting a specific symptom, even if underlying failure types have not changed. This can help detect perception-driven anomalies, such as increased sensitivity to latency or misleading frontend messages (e.g., UI prompts or terminology). Additionally, persistent entropy spikes may inform post-incident reviews by highlighting mismatches between user experience and internal attribution. Tracking such metrics over time supports a more user-aware understanding of reliability.

Service-Specific Resilience Engineering. Based on Observations 14 and 15, distinct reliability patterns across services, such as infrequent but high-impact failures in Character.AI and high user-side availability in DeepSeek, suggest that a one-size-fits-all resilience strategy is suboptimal. Instead, engineering teams can develop service-specific strategies. For instance, fast rollback procedures or preemptive scaling may be more suitable for services prone to rare but severe failures, whereas cost-efficient deployments or lightweight fallback mechanisms may suffice for services with stable operational performance.

4 Temporal Patterns of Failures

This section examines the temporal characteristics of failures in major LLM services using both operator and user reports. The goal is to uncover when failures tend to occur and how temporal patterns differ across services and perspectives. The analysis includes: (1) temporal trends in daily and weekly failure counts; (2) distributions of failures across weekdays, days of the week, and hours of the day; and (3) auto-correlation patterns to evaluate short- and medium-range periodicity in failure occurrences.

4.1 Temporal Trends

Observation #16: *ChatGPT exhibits the most consistent gap between operator and user failure counts, with user reports exceeding operator reports on nearly all days (mean daily gap: 2 failures) and every week (mean weekly gap: 14).*
Observation #17: *Claude shows the opposite pattern: operator-reported failures frequently outnumber user reports, particularly in weekly aggregates (mean weekly gap: 3).*
Observation #18: *Across services, user reports exhibit greater short-term volatility (mean daily variance: 1.30 failures^2), whereas operator reports are more stable and periodic (mean daily variance: 0.87).*

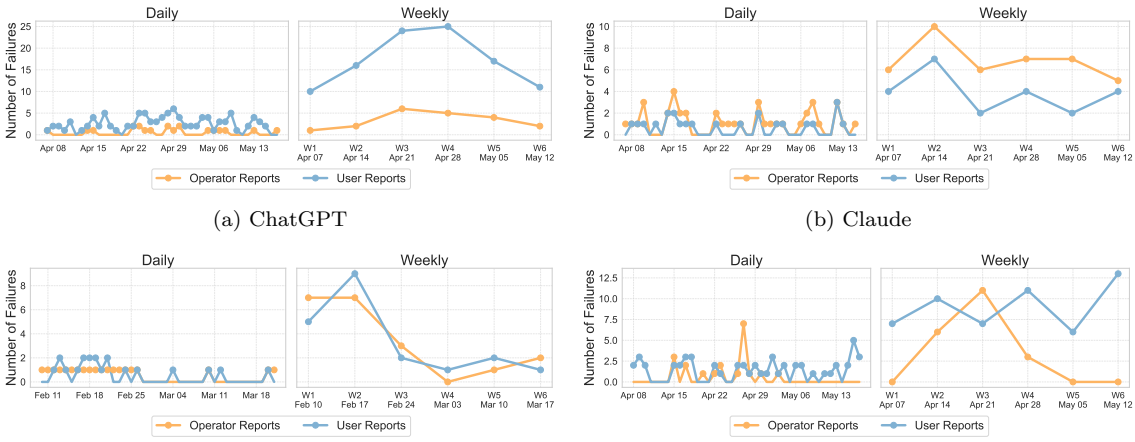
Figure 4.1 illustrates the evolution of failure counts over time for each service, comparing operator and user reports on both daily and weekly scales. Several consistent patterns emerge across services, reflecting fundamental differences in monitoring granularity, reporting mechanisms, and user behavior.

The most persistent discrepancy is observed in ChatGPT. As shown in Fig.4.1a, user reports significantly outnumber operator reports across nearly all days and weeks, with the gap remaining consistently wide over time (Observation16). This suggests that many user-perceived issues are not formally recorded in official reports—either due to localization, transient frontend disruptions, or the operator's reporting threshold. Given ChatGPT's large user base, even small-scale incidents may generate a disproportionate volume of user feedback, amplifying the apparent mismatch between perspectives.

Claude presents a contrasting pattern in Fig.4.1b, where operator-reported failures frequently exceed user reports, especially on the weekly level (Observation17). The relatively low volume and variance in user reports can be partially attributed to Claude's much smaller user base compared to ChatGPT. As of May 2025, Claude receives approximately 3.3 million daily visits [39], whereas ChatGPT sees an estimated 122.6 million daily active users [27]—roughly 37 times higher. This substantial difference in scale naturally results in a far lower volume of user-generated failure feedback for Claude, even if the underlying service reliability were comparable. Consequently, many minor or localized issues may go unreported by users, leading to a flatter and sparser reporting trend from the user side.

A more structural difference emerges when comparing the overall shape of operator and user reporting timelines. Across all services, user reports exhibit sharper fluctuations—spikes and drops occurring over short intervals—whereas operator reports appear more periodic and stable, often following smoother weekly cycles (Observation18). This temporal volatility indicates that user feedback is more reactive and sensitive to visible, real-time disruptions, while operator reports likely result from post-hoc confirmation and

aggregation processes. The result is a temporal desynchronization that reinforces the value of incorporating both perspectives for a complete view of service reliability.



(a) ChatGPT

(b) Claude

(c) DeepSeek

(d) Character.AI

Figure 4.1: Temporal trends in LLM service failures based on operator and user reports, showing daily and weekly failure counts across different service providers.

4.2 Temporal Distributions

Observation #19: *Operator-reported failures exhibit periodic patterns across all services, occurring more frequently on weekdays than on weekends. ChatGPT, Claude, and Character.AI failures peak between 13:00 and 00:00 UTC (06:00–17:00 PDT), while DeepSeek peaks between 01:00 and 10:00 UTC (09:00–18:00 CST), reflecting diurnal patterns that are geographically distinct but consistently aligned with regional working hours.*

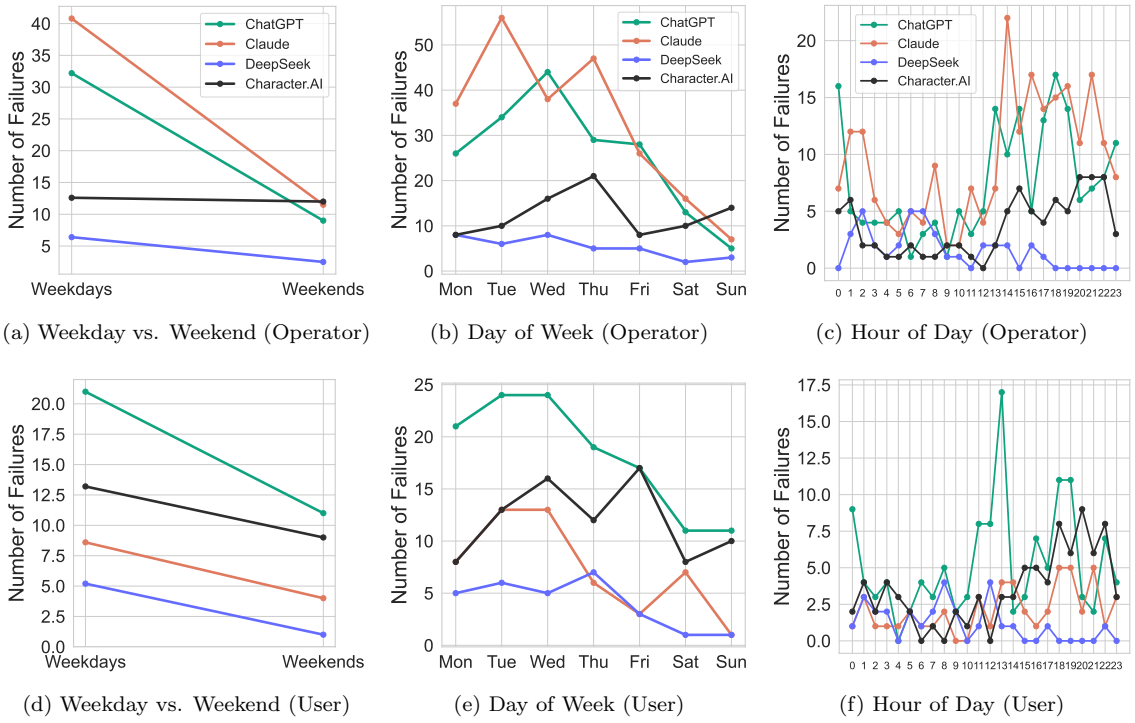
Observation #20: *User-reported failures largely mirror operator-side temporal patterns, showing higher frequencies on weekdays and clear peaks during regional daytime hours. However, short-term hourly variance differs by service: for ChatGPT, user reports fluctuate more sharply than operator reports, whereas the opposite is observed for Claude.*

Figure 4.2 depicts temporal distributions of failure reports across different time resolutions—weekdays versus weekends, days of the week, and hours of the day—captured from both operator and user perspectives. The results reveal consistent periodic patterns as well as subtle differences between reporting sources.

Fig.4.2a shows that operator-reported failures generally occur more frequently on weekdays than weekends across all services (Observation 19). Notably, Character.AI is an exception to this trend, showing almost no difference between weekdays and weekends. However, it still experiences the most failures on Wednesdays and Thursdays (Fig. 4.2b), indicating that incidents remain concentrated within the workweek. Meanwhile, Claude and ChatGPT show broader weekday dominance, with the majority of failures occurring from Monday through Friday. At the hourly level (Fig. 4.2c), all four services display clear diurnal patterns, though with distinct regional signatures. ChatGPT, Claude, and Character.AI failures cluster between 13:00 and 00:00 UTC, corresponding to 06:00–17:00

PDT—typical US working hours, aligning with their primary user base. DeepSeek's failures peak earlier, between 01:00 and 10:00 UTC (i.e., 09:00–18:00 CST), consistent with daytime hours in East Asia. These differences suggest that operator-side incidents are more likely to occur during peak usage periods in different regions, reflecting instability caused by user load or service deployments, which are typically scheduled during local working hours.

User-reported failures broadly mirror these temporal trends. Like operator data, weekday volumes consistently exceed weekend levels across all services (Fig.4.2d), and most user reports occur during local daytime hours (Fig.4.2f). However, a closer examination of short-term hourly variance reveals service-specific differences (Observation 20). For ChatGPT, user-side distributions fluctuate more sharply than operator-side ones, with spikes at 13:00 UTC followed by sudden drops at 14:00 UTC, likely driven by high user sensitivity to transient disruptions. In contrast, Claude displays the opposite pattern: its operator-reported failures show greater temporal variability, while user reports are relatively smooth and consistent. Character.AI and DeepSeek, meanwhile, exhibit similar smoothness from both perspectives. These asymmetries suggest that temporal reporting variance is not a general property of users versus operators, but rather depends on service-specific frontend conditions, monitoring granularity, and user population scale.



(a) Weekday vs. Weekend (Operator) (b) Day of Week (Operator) (c) Hour of Day (Operator)

(d) Weekday vs. Weekend (User) (e) Day of Week (User) (f) Hour of Day (User)

Figure 4.2: Temporal distributions of LLM service failures, comparing operator and user reports. Each row presents three complementary views: weekday vs. weekend, daily distribution across the week, and hourly failure patterns. The failure time is consistently defined as the calendar day, week, and hour of the failure's start time.

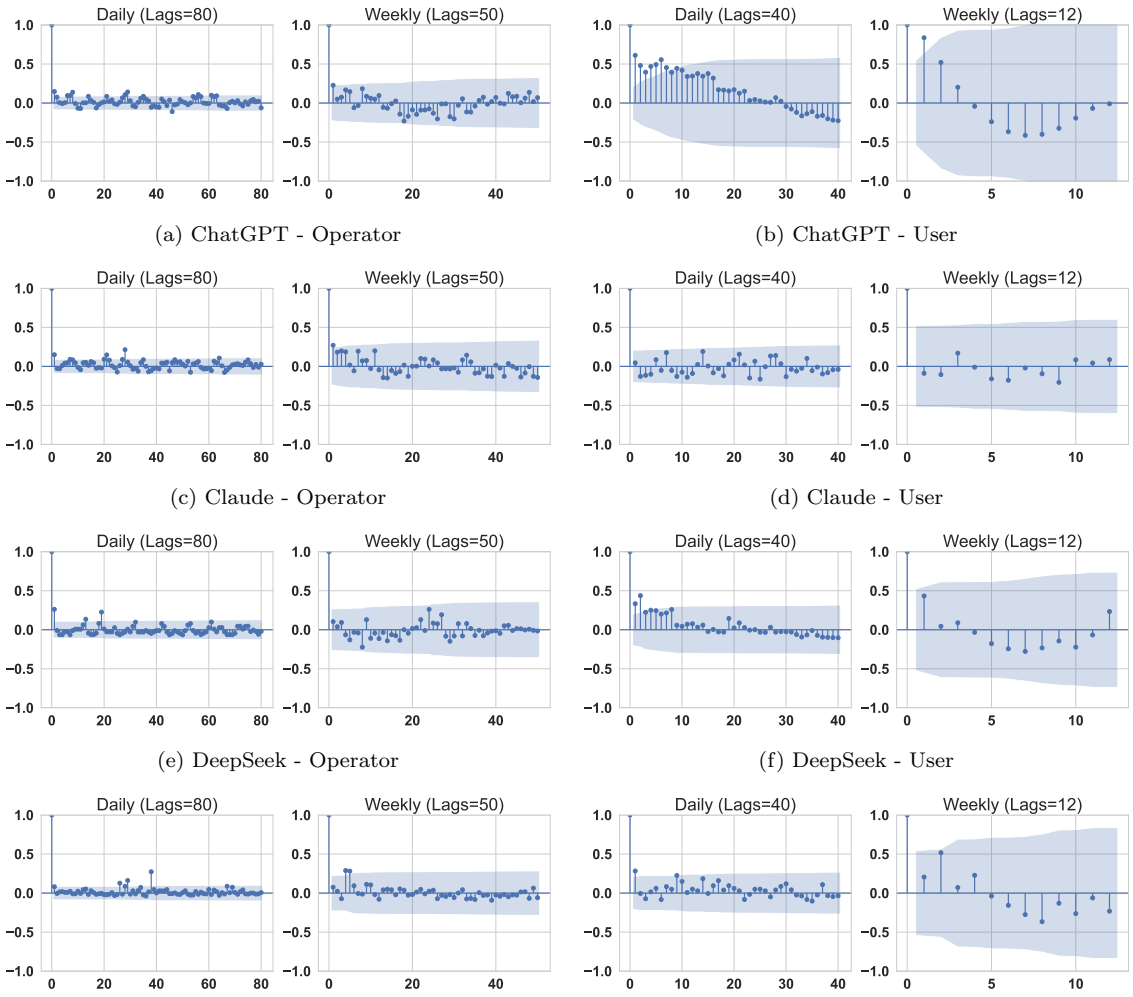
4.3 Auto-correlations

Observation #21: *Operator-reported failures exhibit short-range temporal dependencies with service-specific periodicity. ChatGPT and Claude show clear 1-week periodicity, while Character.AI displays isolated peaks at lags of 4 and 5 weeks, suggesting possible monthly auto-correlations.*
Observation #22: *User-reported failures exhibit weaker periodicity across services. Only ChatGPT shows strong 7-day periodicity, while DeepSeek and Character.AI display short-range daily auto-correlations.*

Figure 4.3 presents the ACFs of operator- and user-reported failure counts, highlighting temporal dependencies such as weekly periodicity. Each subplot combines daily and weekly views to capture correlation structures across short and medium time scales. From the operator perspective, ChatGPT exhibits the most extensive auto-correlations. Its daily ACF (Fig.4.3a) contains significant lags at 1, 6, 7, 8, 28, 29, 46, and 56 days, while the weekly ACF shows a clear peak at lag 1, reflecting a 7-day periodicity. Claude (Fig.4.3c) has significant daily auto-correlations at lag 1, 21, and 28, along with a weekly peak at lag 1. DeepSeek (Fig.4.3e) shows only short-range signals in the daily ACF—at lag 1, 13, and 19—and no significant lags in the weekly ACF. Character.AI (Fig.4.3g) presents three scattered significant daily lags at 26, 29, and 38, along with two isolated peaks at lag 4 and 5 in the weekly ACF, suggesting possible monthly auto-correlations (Observation 21).

From the user side, ChatGPT again stands out with clear temporal structure. Its daily ACF (Fig.4.3b) shows continuous short-range dependencies at lags 1 through 7, and the weekly ACF also has a strong peak at lag 1, indicating a robust 7-day periodicity (Observation22). In contrast, Claude (Fig.4.3d) shows no significant auto-correlation at any lag in either ACF. DeepSeek (Fig.4.3f) has short-term memory only, with significant lags at day 1 and 2, and no weekly auto-correlations. Character.AI (Fig. 4.3h) displays minimal user-side memory, with a single significant daily lag at 1, and no weekly signal.

While several services, particularly ChatGPT, exhibit clear short-term and weekly auto-correlations, the observed periodicity is generally shallow and fragmented, with most significant lags being sparse or isolated. This limits the utility of these patterns for predicting future service failures. Even in the case of ChatGPT, where multiple consecutive lags and weekly signals appear on both operator and user sides, the available signals—though offering some predictive potential—remain weak and unstable, limiting their effectiveness for robust forecasting. Reliable failure prediction would require stronger and more stable indicators, complemented by more advanced modeling approaches.



(a) ChatGPT - Operator

(b) ChatGPT - User

(c) Claude - Operator

(d) Claude - User

(e) DeepSeek - Operator

(f) DeepSeek - User

(g) Character.AI - Operator

(h) Character.AI - User

Figure 4.3: Auto-correlations with the numbers of failures aggregated at different time granularities, based on operator and user reports. The shaded blue area indicates the 95% confidence interval; points outside this band suggest statistically significant correlations.

4.4 Summary of Temporal Patterns

This section addresses RQ3 by analyzing temporal patterns in failure occurrences across services and sources. The analysis focuses on inter-source discrepancies, time-of-day and day-of-week variations, and periodic structures in both operator and user-reported failures.

4.4.1 Key Observations

ChatGPT exhibits the most consistent inter-source discrepancy, with user-reported failures exceeding operator counts nearly every day (mean daily gap: 2 failures) and every week (mean weekly gap: 14) (Observation 16). In contrast, Claude shows the opposite pattern, where operator-reported failures often outnumber user reports (mean weekly gap: 3) (Observation 17). Across all services, user reports exhibit greater short-term volatil-

ity in failure counts, with a mean daily variance of 1.30 failures^2, compared to 0.87 for operator reports (Observation 18).

Operator reports show a consistent weekday–weekend cycle, with failures occurring more frequently on weekdays than on weekends, along with service-specific diurnal patterns. ChatGPT, Claude, and Character.AI failures peak between 06:00 and 17:00 PDT, while DeepSeek failures peak between 09:00 and 18:00 CST. These peaks align with typical daytime hours in each service's primary user regions (Observation 19). User reports largely mirror these temporal distributions, also peaking during local working hours. However, the magnitude of hourly fluctuations varies: ChatGPT user reports are more volatile than operator reports, whereas Claude shows the reverse pattern (Observation 20).

Periodicity analysis reveals short-range dependencies in operator data. ChatGPT and Claude display clear 7-day cycles, while Character.AI shows longer periodicity with peaks at lags of 4 and 5 weeks (Observation 21). User-side periodicity is less pronounced: only ChatGPT maintains a strong 7-day cycle, while DeepSeek and Character.AI exhibit weaker short-range daily dependencies (Observation 22).

4.4.2 Use Cases Derived from Observations

Monitoring Persistent Reporting Gaps for Reliability Blind Spots. Based on Observations 16 and 17, services like ChatGPT and Claude exhibit persistent and directionally consistent discrepancies in daily and weekly failure counts between operator and user reports. These sustained gaps may reflect under-reporting or monitoring blind spots on one side. Providers can track these gaps over time as reliability signals. Internal reviews can be triggered when the discrepancies exceed service-specific thresholds or persist across multiple days or weeks. Gap magnitudes can be quantified and visualized in reliability dashboards, or integrated into alerting systems that automatically flag abnormal patterns when they deviate from historical baselines.

Volatility-Aware Alert Thresholds. Observation 18 shows that user-reported failure volumes are more volatile than operator-reported ones on short timescales. This discrepancy suggests that uniform alerting thresholds may not suit both perspectives. For example, alert systems based on user data may require adaptive smoothing or volatility-aware thresholding to avoid false positives during normal fluctuations. Conversely, operator-side monitoring can prioritize trend detection over spike sensitivity. Designing alert logic that respects source-specific volatility improves responsiveness while reducing noise.

Temporal Modeling for Proactive Failure Detection. Based on Observations 19 and 20, the clear periodic and diurnal patterns observed in both operator- and user-reported failures provide a foundation for time-aware failure prediction. Historical failure volumes can be used to train temporal models that highlight high-risk time windows for each service. These models enable proactive mitigation by performing stricter pre-deployment checks, temporarily increasing system redundancy, or lowering alert thresholds to detect issues more quickly during periods of elevated risk. Such time-aware actions help improve preparedness and reduce downtime.

5 Correlation Analysis of Failures

This section investigates correlations between failures across services and perspectives to evaluate the extent of alignment and interdependence. The analysis includes: (1) the temporal correlation between user-reported issue volumes and official incident acknowledgements; and (2) the co-occurrence of failures across services, from both operator and user perspectives.

5.1 Correlation Between Operator and User Reported Failures

Observation #23: *All services exhibit statistically significant correlations between 5-minute user-reported issue volumes and official incidents, although the overall strength of alignment is weak and varies across services (Pearson: 0.060–0.315; Spearman: 0.053–0.147). Among the services, Character.AI shows the largest discrepancy (0.203) between its Pearson (0.315) and Spearman (0.112) coefficients.*

Table 5.1 presents Pearson and Spearman correlation coefficients between user-reported issue volumes and the timing of official incident acknowledgements. All four services exhibit statistically significant correlations ($p < 0.001$), confirming that user reporting activity is meaningfully aligned with periods of operator-confirmed incidents (Observation 23).

Despite the statistical significance, the absolute strength of correlation varies considerably. Character.AI shows the highest Pearson correlation (0.315), suggesting that user activity is more strongly synchronized with official incident timelines for this service. ChatGPT and Claude follow with moderate correlations (0.192 and 0.162 respectively), while DeepSeek stands out with the weakest relationship (0.060), indicating a limited coupling between user-reported issue volumes and official incidents.

The Spearman correlations follow a similar pattern, though values are slightly lower overall. Notably, Character.AI shows a steep drop from Pearson (0.315) to Spearman (0.112) (Observation 23), suggesting that this alignment is not consistent over time. The correlation is likely inflated by a few days when user reports spiked dramatically and happened to coincide with operator-reported failures. On most days, however, the relationship appears weak or unstable, indicating that the observed correlation is driven by rare, extreme events rather than a steady underlying pattern. In contrast, the other services show only minor differences between Pearson and Spearman values, suggesting that their alignment is less dependent on outliers and more reflective of persistent trends.

These results indicate that while user volumes can serve as a useful indicator for official incidents, the reliability of this signal varies by service. In particular, the weak correlations observed for DeepSeek highlight the need for service-specific calibration when integrating user reporting into real-time failure detection or alerting systems.

Table 5.1: Pearson and Spearman correlation coefficients between user-reported issue volumes (aggregated in 5-minute bins) and official incident acknowledgements for each LLM service. Asterisks denote statistical significance: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

Service	**Pearson Correlation**	**Spearman Correlation**
ChatGPT	0.192^{***}	0.139^{***}
Claude	0.162^{***}	0.147^{***}
DeepSeek	0.060^{***}	0.053^{***}
Character.AI	0.315^{***}	0.112^{***}

5.2 Co-occurrence of Failures

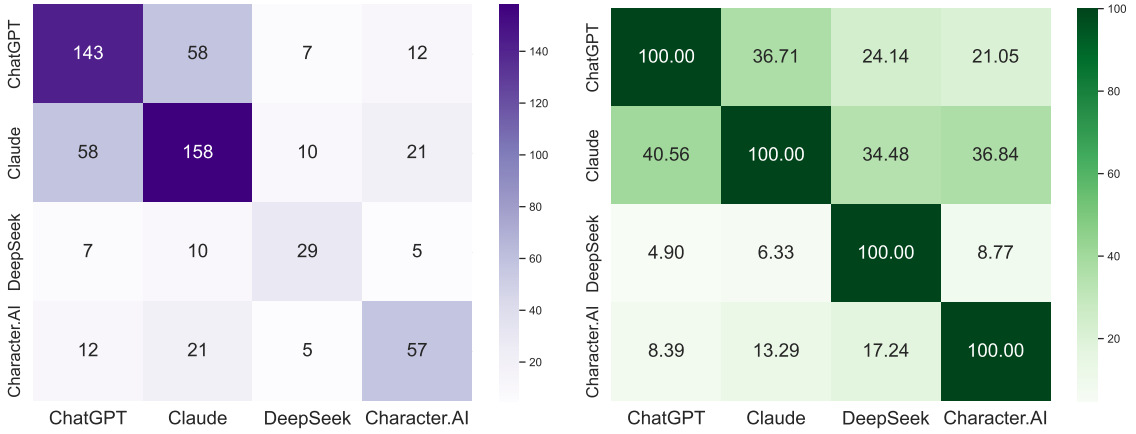
Observation #24: *On the operator side, failure co-occurrence is most prominent between ChatGPT and Claude, which share 58 incident days and exhibit elevated conditional probabilities (40.56% and 36.71%). In contrast, DeepSeek shows minimal co-occurrence with other services, indicating a high degree of operational independence.*
Observation #25: *User-reported failures reveal strong cross-service co-occurrence, especially between ChatGPT and Character.AI, which share 34 failure days and exhibit mutual conditional probabilities exceeding 60%. While potential shared infrastructure may contribute, the large and active user bases of both services likely increase the likelihood of detecting and reporting even minor degradations on the same day.*

To evaluate potential dependencies between LLM services, Figure 5.1 presents co-occurrence matrices derived from both operator-reported and user-reported failures. Specifically, the analysis examines the number of shared failure days and the conditional probability that one service experiences a failure given that another service fails on the same day.

On the operator side, the most notable co-occurrence is observed between ChatGPT and Claude (Observation 24). These two services share 58 incident days (Fig.5.1a), and their conditional failure probabilities reach 40.56% (Claude given ChatGPT) and 36.71% (ChatGPT given Claude), respectively (Fig.5.1b). No other service pair shows comparable values. In contrast, DeepSeek shares only 7 incident days with ChatGPT and 10 with Claude, and the conditional probability of DeepSeek failing given an incident in ChatGPT or Claude remains low (4.90% and 6.33%, respectively), suggesting a high degree of operational independence. Although the reverse is substantially higher—24.14% (ChatGPT given DeepSeek) and 34.48% (Claude given DeepSeek)—this merely reflects the rarity of DeepSeek failures rather than any strong mutual dependency.

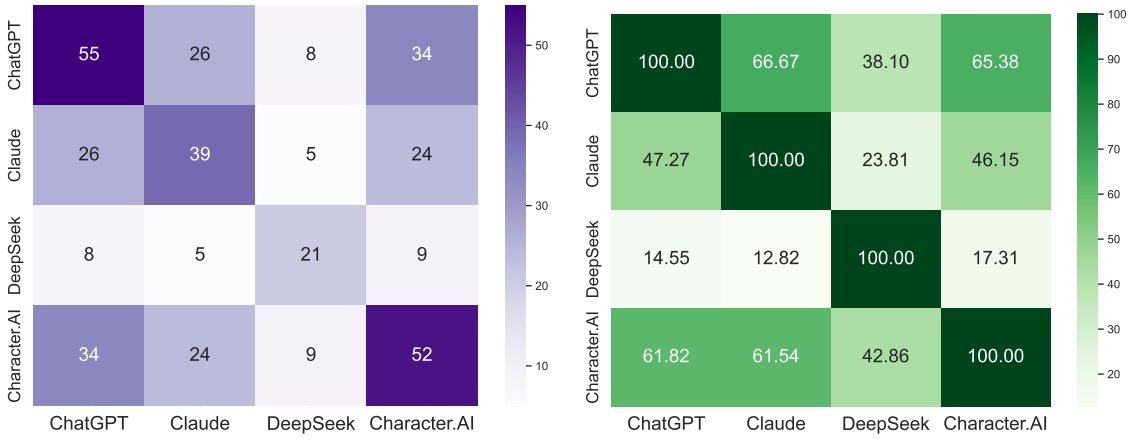
User-reported failures reveal a different pattern. ChatGPT and Character.AI share 34 user failure days (Fig.5.1c), with 61.82% of ChatGPT's user failures and 65.38% of Character.AI's co-occurring with the other (Fig.5.1d), forming the strongest user-side alignment (Observation 25). This strong co-occurrence likely reflects the large and highly active user bases of both services, which increase the chance of detecting and reporting even minor degradations, leading to more frequent overlap in reported failures. Another notable pair is Claude and Character.AI, which share 24 co-occurring days, with conditional probabilities of 61.54% (Character.AI given Claude) and 46.15% (Claude given Character.AI). Additionally, ChatGPT and Claude—already the most aligned pair in operator reports—show

strong user-side co-occurrence as well, with 26 shared user failure days and conditional probabilities of 66.67% and 47.27%. This cross-source consistency may suggest underlying infrastructure-level coupling or shared service dependencies between the two providers. However, as the conditional probabilities on both sides remain in the 40–60% range, these results indicate possible but not definitive operational entanglement.



(a) Co-occurrence failures in days count based on operator reports.

(b) Conditional probabilities (%) of co-occurrence failures based on operator reports. Notes: y-axis = service A, x-axis = service B, cells = $P(A \mid B)$.



(c) Co-occurrence failures in days count based on user reports.

(d) Conditional probabilities (%) of co-occurrence failures based on user reports. Notes: y-axis = service A, x-axis = service B, cells = $P(A \mid B)$.

Figure 5.1: Co-occurrence of failures across LLM services based on operator and user reports. Left: number of co-occurring failures in the same day. Right: conditional probability of one service failing given another fails.

5.3 Summary of Correlation Analysis

This section addresses RQ4 by analyzing both the statistical correlation between the presence of operator-reported incidents and user-reported issue volumes, and the co-occurrence of failures across services.

5.3.1 Key Observations

All services exhibit statistically significant correlations between user-reported issue volumes and official incidents. However, the overall strength of alignment is weak and varies across services. Pearson correlation coefficients range from 0.060 to 0.315, and Spearman from 0.053 to 0.147 (Observation 23). While other services show similar Pearson and Spearman values, Character.AI displays the largest discrepancy (0.203), with a Pearson coefficient of 0.315 and a Spearman of 0.112. This suggests that the alignment may be driven by a few large spikes in user volume rather than consistent temporal patterns.

On the operator side, failure co-occurrence is most notable between ChatGPT and Claude, which share 58 failure days and exhibit elevated conditional probabilities of 40.56% and 36.71% respectively (Observation 24). This pattern may reflect shared infrastructure or synchronized deployment events. In contrast, DeepSeek shows minimal overlap with other services, indicating greater operational independence.

From the user perspective, cross-service co-occurrence is even stronger. ChatGPT and Character.AI share 34 user-reported failure days, with mutual conditional probabilities exceeding 60% (Observation 25). Besides potential infrastructure interdependencies, a more observable explanation based on the user reports is the large and active user bases of both services, which increase the likelihood of failure reports. With higher reporting frequency, the chance of failures being recorded on the same day across services also rises, including minor disruptions that may not be formally acknowledged by operators.

5.3.2 Use Cases Derived from Observations

Using Cross-Service Failure Dependencies for Operational Decision-Making. Based on Observations 24 and 25, ChatGPT, Claude, and Character.AI exhibit substantial cross-service failure co-occurrence, though with different source perspectives: operator-side for ChatGPT and Claude, and user-side for ChatGPT and Character.AI. These patterns suggest that some services may be partially interdependent or simultaneously exposed to shared risks, such as common infrastructure components or synchronized deployments. Providers can monitor co-occurrence rates to detect such hidden dependencies and support more informed operational planning. For instance, when failure co-occurrence is high, LLM providers can coordinate release schedules or infrastructure changes across the services they operate to reduce the risk of cascading disruptions. They can also establish fallback or isolation mechanisms to contain the impact of failures when interdependencies are present.

6 Consistency Analysis Between Sources

This section evaluates the consistency between operator and user reports, aiming to assess the extent to which the two perspectives agree on both failure types and timelines. The analysis includes: (1) the alignment of failure type classifications between sources; and (2) the temporal consistency of reported failure periods.

6.1 Consistency of Failure Types

Observation #26: *Claude exhibits the highest failure type match rate (81.8%), reflecting strong alignment between operator- and user-reported failure types. In contrast, Character.AI shows the lowest match rate (6.5%), primarily because most of its operator reports do not specify a concrete failure type and are therefore labeled as Unknown.*
Observation #27: *The most frequent mismatch occurs when operator-reported failures are misclassified by users as Error Received, which accounts for an average of 62.7% of all misclassified cases across services. This suggests that user reports tend to be symptom-based rather than cause-based, or that certain failures inherently manifest through multiple observable symptoms.*

To evaluate the consistency of failure types between two sources, Fig.6.1 presents a comparison based on the most frequent user-reported failure type within each operator-defined failure window. Fig.6.1a summarizes the overall match rate across services, while Fig. 6.1b provides a detailed breakdown of operator–user failure type mappings.

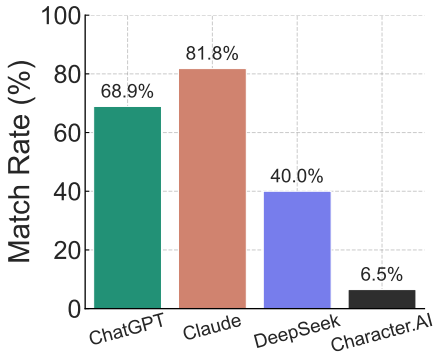
Match rates vary substantially across services. Claude exhibits the highest degree of consistency, with an 81.8% match rate between operator- and user-side labels (Observation 26). This alignment is further supported by the joint distribution matrix (Fig.6.1b), where a majority of failures are classified as *Error Received* by both sides. In contrast, Character.AI shows the lowest match rate (6.5%), as the majority of its operator reports contain only vague descriptions such as *Investigating an issue*, resulting in failure types labeled as *Unknown* (Observation 26). Meanwhile, users frequently label the same incidents as *Inaccessible* or *Slow*, which are clearly more specific. This highlights the limited diagnostic information provided in Character.AI's official incident reports, which hinders category-level alignment with user perceptions.

ChatGPT also shows a relatively high match rate of 68.9%. Given a sufficient number of operator-reported incidents and the substantial volume of user reports, this figure provides a stable and reliable estimate of cross-source consistency.

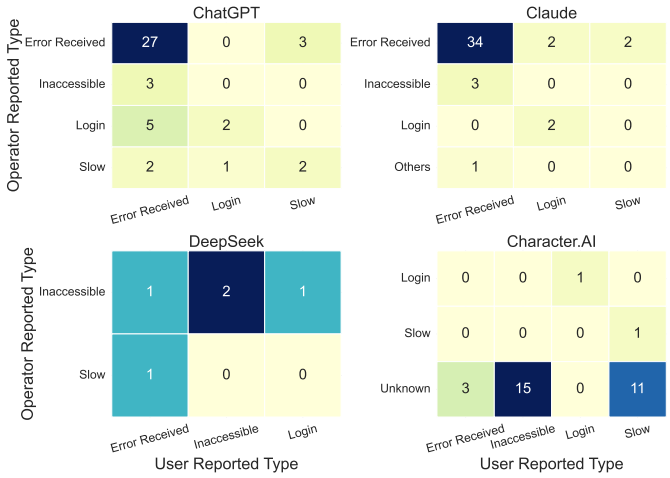
The match rate for DeepSeek is 40.0%, which is relatively low compared to other services. However, this figure should be interpreted with caution due to the small sample size. Only five failures were officially recorded for DeepSeek during the 13-week data collection period, of which two were aligned with the dominant user-reported failure type. While the result is numerically accurate, it may not reliably reflect broader consistency patterns.

Beyond overall alignment, common patterns of mismatch also emerge. In particular, users often classify operator-reported failures—such as *Login* or *Slow*—as *Error Received*, making it the most frequent user-side label across services (Fig.6.1b). This suggests that user reports tend to reflect observable symptoms (e.g., error messages) rather than underlying causes, or simply default to the most common category without distinguishing

between more specific failure types. Another source of mismatch involves the frequent confusion between *Error Received* and *Slow*, which may occur when operator-reported failures simultaneously involve both latency and elevated error rates [30, 28]. Depending on user's interaction context, the same underlying issue may be perceived as either a delay or an explicit error, leading to divergent user classifications (Observation27).



(a) Bar chart of match rates for each LLM service, reflecting the proportion of operator-reported failures whose failure types match the most frequently reported user failure types in the same time window.



(b) Heatmaps showing the joint distribution of operator- and user-reported failure types for each LLM service.

Figure 6.1: Consistency of failure types between operator and user reports across LLM services, evaluated through match rates and joint distributions.

6.2 Consistency of Failure Periods

Observation #28: *ChatGPT and Claude show relatively strong alignment between operator and user reports, though for different reasons: ChatGPT due to high operator coverage (82.22%), and Claude due to accurate user reporting (74.00%) and closely aligned user detection (median lead time: 3 minutes). In contrast, DeepSeek and Character.AI exhibit weaker consistency, with lower overlap rates between the two sources.*

Table 6.1 summarizes the consistency-related statistics between operator- and user-reported failures across services. Each metric in the table captures a specific aspect of this cross-source alignment. The first three metrics report the number of operator-reported incidents (*# of Operator Reports*), the number of user-reported failure periods (*# of User Reports*), and the number of temporally overlapping periods (*# of Overlapping Reports*). The next two metrics—*Coverage of Operator* and *Coverage of User*—measure the proportion of operator and user reports, respectively, that have an overlapping period in the other source. The final two metrics, *Mean User Lead Time* and *Median User Lead Time*, quantify the average and median number of minutes by which user-reported failures precede (or lag behind) the corresponding operator reports. These metrics jointly characterize not only the extent of agreement between sources, but also the temporal sensitivity and reporting behavior of each.

Table 6.1: Consistency-related metrics summarizing the overlap and timing alignment between operator and user reports. Legend: m = minute(s).

Metric	ChatGPT	Claude	DeepSeek	Character.AI
# of Operator Reports	45	74	4	40
# of User Reports	127	50	12	83
# of Overlapping Reports	37	37	2	20
Coverage of Operator [%]	**82.22**	50.00	50.00	50.00
Coverage of User [%]	29.13	**74.00**	16.67	24.10
Mean User Lead Time [m]	72.86	-1.12	14.71	78.13
Median User Lead Time [m]	31.55	**3.00**	14.71	43.50

For ChatGPT (Fig. 6.2), the consistency between sources is relatively high (Observation28). Among the 45 operator-reported incidents, 82.2% overlap with at least one user-perceived failure period. Despite the user reports being more numerous (127 in total), only 29.1% correspond to officially reported incidents. This asymmetry suggests that while most operator-reported failures are noticeable to users, a large portion of user-reported issues remain unacknowledged by the provider. This may be primarily due to users' heightened sensitivity to degraded performance, which may not always meet the operator's threshold for a status update. On average, users report failures 73 minutes earlier than the corresponding official acknowledgment, with a median lead time of 31.6 minutes. This indicates that ChatGPT users often act as early detectors of emerging issues.

Further analysis of the overlapping incident descriptions reveals that operator-reported failures also perceived by users often involve highly visible and widely used components such as *Web*, *Login*, *Search*, and *File uploads*. These failures are more likely to impact a broad portion of the user base and therefore generate substantial user feedback. In contrast, operator-reported incidents not detected by users often involve more specialized or less commonly used components. These include outages in legacy or experimental models such as *o3-mini*, failures affecting only enterprise users, disruptions in *Temporary chat* (primarily used by free users), or issues with *ChatGPT Voice*. What these failures share is limited visibility—they do not affect the general user population and, as a result, are less likely to be reflected in user reports.

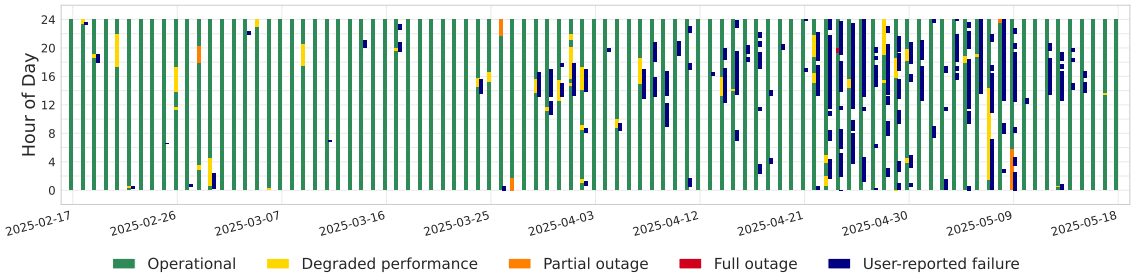


Figure 6.2: Failure periods reported by operators versus users for ChatGPT, plotted by date and hour of the day. Different colors indicate operator-reported impact levels and user-reported failures.

Claude (Fig. 6.3) exhibits a distinct pattern. It has more operator-reported incidents (74) than user-perceived failures (50), yet 74.0% of user reports overlap with official incidents. This suggests that Claude users are relatively accurate in detecting provider-acknowledged failures (Observation28). However, only 50.0% of operator-reported incidents are reflected in user reports, which may be attributed to the overall low reporting volume—likely a result of Claude's relatively small user base and the fact that most users primarily interact with the default, up-to-date chat models, such as *Claude 3.7 Sonnet*.

This interpretation is supported by the content of the overlapping incidents, which largely involve core functionalities or widely used components. In contrast, many unmatched operator-reported incidents pertain to more specialized features, such as *long context requests*, issues with receiving *magic login links*, or failures in optional backup models available via the web interface, such as *Claude 3.5 Haiku*. These components are less likely to be used by the general population and thus may not trigger sufficient user reports. Furthermore, due to the much smaller user base, some unmatched operator-reported incidents even involve the default model, *Claude 3.7 Sonnet*. In such cases, elevated error rates were observed in the model. However, if only a small number of users were active at the time—or if the disruption was short-lived—there may not have been sufficient user feedback to form a user-reported failure period.

The mean user lead time is nearly zero at -1.1 minutes, with a median of +3 minutes, indicating near-simultaneous perception of failures between users and the provider.

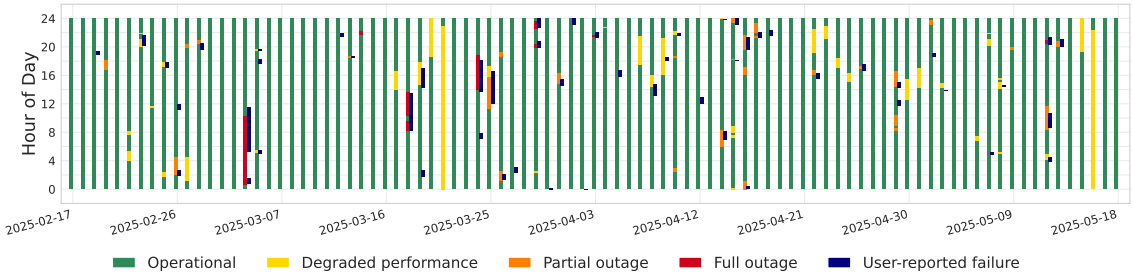


Figure 6.3: Failure periods reported by operators versus users for Claude.

For DeepSeek, the number of reports is much smaller: only 4 operator-reported incidents and 12 user-reported failures during the observation period (excluding one outlier incident that spanned 19 days, from 2025-02-08 to 2025-02-26, and overlapped with multiple user-reported failures, as shown in Fig. 6.4). Half of the official incidents (2) were matched by user-reported failure periods, while only 16.7% of user reports correspond to officially acknowledged incidents (Observation28). This low user-side coverage may be partially explained by the fact that users often report issues upon receiving a *Server is busy* message, which the provider usually does not acknowledge as a formal incident, as this is a common occurrence for open-source models under high demand.

The mean and median lead times are both 14.7 minutes, suggesting a mild tendency for users to detect issues earlier—though the small sample size limits the generalizability of this pattern. Furthermore, all operator incident descriptions are nearly identical, simply stating *DeepSeek Web/API Service Not Available*, leaving no room to infer component-level patterns from either matched or unmatched incidents.

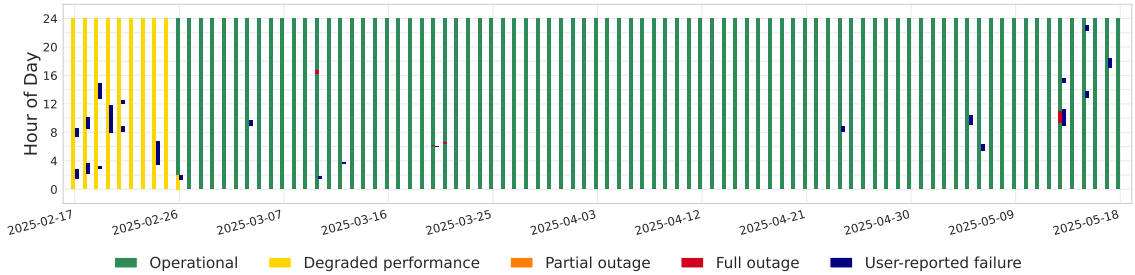


Figure 6.4: Failure periods reported by operators versus users for DeepSeek.

Character.AI (Fig. 6.5) demonstrates moderate misalignment. Of its 40 operator-reported incidents, only 50.0% were picked up by users, while just 24.1% of the 83 user-reported failures matched operator data (Observation28). The user lead time is substantial: on average, users report failures 78.1 minutes earlier than the corresponding operator reports, with a median lead time of 43.5 minutes. This suggests either delayed response from the provider or stricter thresholds for acknowledging an incident.

Most operator reports adopt vague phrasing such as "Investigating an issue," offering little detail. Still, a few operator-reported incidents that were detected by users contain clear descriptions, referring to *general degradation* or *login issues*. These types of failures are more likely to prompt widespread user response. Since no consistent pattern can be discerned from the operator side, unmatched user reports were further examined and found to be dominated by *Slow* (66%) and *Inaccessible* (24%) types, indicating high user sensitivity to service latency and availability.

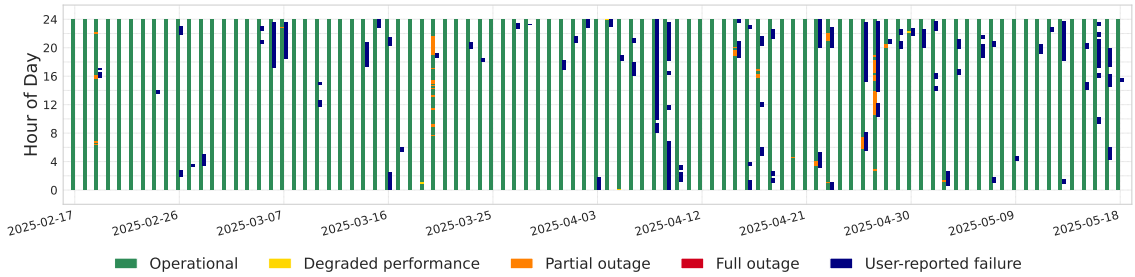


Figure 6.5: Failure periods reported by operators versus users for Character.AI.

6.3 Summary of Consistency Analysis

This section addresses RQ5 by evaluating the consistency between operator and user reports in terms of failure types and temporal alignment of failures. The analysis reveals service-specific differences in how closely user reports align with official incident descriptions and timelines.

6.3.1 Key Observations

Claude demonstrates the strongest agreement in failure type classification, with a match rate of 81.8% between operator- and user-reported failure types. In contrast, Character.AI shows the lowest match rate (6.5%), primarily due to its frequent lack of detailed descriptions in official incident reports, leading to failure types being extracted as *Unknown* (Observation 26). Across services, the most common misclassification involves users labeling operator-reported failures as *Error Received*, which accounts for 62.7% of all mismatches. This pattern suggests that users tend to report based on perceived symptoms rather than root causes, or that certain failures inherently manifest through multiple observable symptoms (Observation 27).

Beyond failure classification, source alignment also varies in terms of temporal overlap and coverage. ChatGPT and Claude demonstrate relatively high cross-source consistency, though for different reasons (Observation 28). ChatGPT benefits from a high proportion of overlapping failure periods relative to the number of operator-reported incidents (82.22%) and early user detection, with a median lead time of 31.55 minutes. In contrast, Claude achieves strong alignment primarily through accurate user reporting, as reflected by a high overlap rate relative to the number of user reports (74.00%). DeepSeek and Character.AI, however, exhibit weaker consistency between sources (Observation 28), with fewer overlapping failure periods and lower agreement in failure type classification.

6.3.2 Use Cases Derived from Observations

Leveraging Consistency-Related Metrics for Reliability Strategy Design. Based on Observation 28, alignment between operator and user reports varies across services. For services with high operator coverage like ChatGPT, most operator-reported failures are also perceived by users, indicating that the incidents captured by internal systems largely reflect user-visible issues. This alignment makes it more feasible to automate incident workflows, such as updating real-time dashboards, syncing status via APIs, or publishing reports to official status pages, without requiring additional user-side confirmation. In contrast, for services with high user coverage like Claude, user feedback can serve as a reliable external signal to complement internal monitoring, especially when certain failures go undetected by operators.

Lead time characteristics further enhance these strategies. When user reports significantly precede official acknowledgements (e.g., 31.55 minutes for ChatGPT), this early-warning capacity can inform provisional alerts or preemptive traffic rerouting. When user detection closely aligns with operator timing (e.g., 3-minute lead in Claude), user feedback can serve as a confirmation layer to validate internal detection and reduce false positives.

7 Threats To Validity

This section outlines key threats to the validity of this study, categorized according to the characteristics and limitations of the data sources used, the methods employed for failure classification, and the overall scope of the analysis.

Threats to Data Completeness The 24-hour user report monitoring script was deployed on a GCP instance to continuously collect data from DownForEveryoneOrJustMe. However, GCP instances may be vulnerable to bot attacks or unexpected downtime. In the event of such disruptions, a temporal gap can arise between the onset of the downtime and the recovery, during which user-reported failures may be missed. As a result, the collected user report data cannot be guaranteed to be 100% complete.

Threats to Label Accuracy from LLM-based Extraction This study relies on LLMs such as ChatGPT-4o to semantically classify failure types from semi-structured operator reports and structured user reports containing text columns. Although the classification prompts were designed in a zero-shot or few-shot format with detailed instructions and consistent schemas, the LLM-based processing remains inherently opaque. There is no deterministic guarantee that the inferred labels fully capture the intended semantics, particularly in ambiguous cases. Furthermore, since this approach operates as a black box, verifying classification correctness is difficult without extensive human validation. To partially mitigate this, manual spot-checking was performed on sampled outputs; however, full-scale verification remains infeasible given the dataset size.

Threats from Operator Report Inconsistencies Operator-provided incident reports sometimes suffer from imprecision and inconsistency, which pose challenges for downstream failure analysis. First, several reports do not specify a time zone [4], making it difficult to determine the precise start and end times of an incident. Second, in some cases, the reported date or time is factually incorrect [6]. Third, failure type descriptions are sometimes vague, or even not specified. For example, some reports only use ambiguous titles such as *Errors for logins and requests* [3, 5], while others—especially those from Character.AI—provide no meaningful description at all and contain only generic phrases like *Investigating an issue* [9]. Finally, inconsistencies are occasionally observed between the `timeline` field and the free-text write-up of an incident. In particular, OpenAI's status page sometimes presents conflicting start and end times between these two fields, requiring manual adjudication [31].

Threats from User Data Ambiguity User reports are inherently noisy, informal, and may reflect localized network problems or client-side issues rather than actual service-wide failures. Although semantic classification via LLMs was employed to extract structured failure types, the original user-generated content remains subjective and occasionally ambiguous. Additionally, third-party platforms like DownForEveryoneOrJustMe only retain the most recent 20 reports, limiting the recoverability of historical data.

Threats from Scope Limitation This study focuses exclusively on the reliability of web-based chat services and does not include other components such as API services. For

example, API-related issues are not covered due to the limited availability of user-reported failures that specifically reference API usage, as discussed in Section 2.1. The sparsity of such data makes it difficult to conduct meaningful failure pattern analysis at the API level. As a result, the findings presented in this work pertain only to the end-user-facing chat services of major LLM providers and may not generalize to other service components offered by these providers.

8 Related Work

8.1 LLM Workloads

Recent advances in LLMs have led to diverse usage scenarios across pretraining, adaptation, and inference. Each of these stages introduces distinct workload patterns with varying computational demands, scheduling behaviors, and resource footprints. Understanding these workload characteristics has become essential for optimizing LLM system design, benchmarking, and operational management. This section reviews key studies that characterize LLM workloads across different stages of the model lifecycle, from foundation model training to production-level inference and workload modeling.

Training Workloads Large-scale training of foundational language models demands extensive computational resources, advanced parallelization techniques, and scalable distributed infrastructure. Prior studies have examined how these demands manifest as system-level workloads. A recent survey by Duan et al. [15] explores strategies for efficient training across multiple nodes and GPUs, including pipeline parallelism, tensor parallelism, and data parallelism, while addressing bottlenecks such as communication overhead and memory constraints. These training workloads are characterized by high GPU utilization, long job durations, and coordinated orchestration across clusters, forming the foundation of LLM workload analysis.

Adaptation Workloads Beyond pretraining, modern LLM applications frequently involve adaptation workloads such as fine-tuning, instruction tuning, or retrieval-augmented generation (RAG). These workloads introduce new system behaviors including irregular batch sizes, varied context lengths, and frequent data I/O for external knowledge integration. For example, Lewis et al. [23] introduce RAG as a hybrid architecture combining LLMs with document retrieval, resulting in increased latency variability and I/O-bound patterns. Additionally, recent studies on parameter-efficient tuning (e.g., LoRA [20] and adapters [19]) suggest a shift toward more lightweight and fine-grained tuning workloads with lower memory demands but higher variability. These adaptation phases are essential to modeling realistic and emerging LLM use cases in production systems.

Inference Workloads LLM inference workloads pose distinct challenges related to latency, throughput, and GPU memory fragmentation, especially in multi-tenant or real-time serving environments. Vellaisamy et al. [36] systematically evaluate inference workloads on CPU-GPU coupled architectures, highlighting the performance implications of interconnect bandwidth, memory swapping, and kernel-level execution delays. Complementing this, Chitty-Venkata et al. [10] introduce LLM-Inference-Bench, a comprehensive benchmarking suite that evaluates the inference performance of various LLMs across multiple AI accelerators and inference frameworks, such as vLLM, TensorRT-LLM, DeepSpeed-MII, and llama.cpp. Their study provides detailed insights into the trade-offs between latency and throughput under different batching and scheduling strategies, contributing to a better understanding of inference workload characteristics.

Real-world LLM Workload Modeling Recent research has shifted toward holistic workload characterization, combining multiple LLM usage stages and real production traces. Hu et al. [21] analyze GPU traces over six months from a large-scale AI lab, uncovering temporal and resource usage patterns in real-world LLM development. Building upon this, Xiang et al. [41] propose a data-driven workload generator based on production traffic traces, capable of replicating realistic model serving scenarios across diverse task types. These contributions enable workload modeling beyond synthetic benchmarks and lay the foundation for reproducible system evaluation, scheduling policy design, and capacity planning in LLM deployment.

Taken together, these studies provide a comprehensive view of LLM workloads as they manifest across training, adaptation, inference, and real-world deployment. By identifying workload-specific performance bottlenecks, temporal dynamics, and system constraints, prior research has laid a solid foundation for downstream investigations into LLM service reliability. In particular, the characterization of workload heterogeneity and operational complexity directly informs this study's analysis of failure patterns, offering critical context for understanding the underlying causes and manifestations of service-level disruptions in LLM systems.

8.2 Failure Characterization

Failure characterization spans multiple layers of LLM system architecture, from underlying hardware components to user-facing services. At the hardware layer, several studies have investigated reliability issues in large-scale HPC infrastructures. For example, Cui et al. [13] provide a fine-grained characterization of GPU failures by analyzing two and a half years of error logs in a large-scale AI/HPC system. The study identifies the GPU System Processor (GSP) as the most failure-prone components, and challenges conventional assumptions by demonstrating that GPU memory is significantly more reliable than other GPU hardware components. In parallel, George et al. [17] characterize large-scale disk failures in HPC environments using over 5,000 failure records, analyzing them across temporal, spatial, and relational dimensions to inform storage system reliability. Chu et al. [11] further explore the operational characteristics of ML jobs on HPC clusters, showing that they tend to produce higher failure rates, longer runtime, and greater thermal stress compared to generic jobs. Together, these studies establish foundational insights into system-level failure behavior under intensive LLM workloads.

Moving up to the application layer, failure analysis has increasingly focused on observable behaviors and failure types in user-facing software systems. Anandayuvaraj et al. [1] propose a novel LLM-based pipeline that extracts and summarizes software failure events from news sources, offering an automated view of how failures manifest in public reports. Tie et al. [35] conduct an empirical user study in web development contexts, identifying nine categories of LLM failures grounded in real-world software engineering tasks and user interactions. Talluri et al. [34] characterize failures in cloud, web, and gaming services by collecting and analyzing reports from both users and operators. Their multi-source dataset enables a cross-layer analysis of service reliability, revealing that high-level user facing services exhibit fewer failures than low-level infrastructure services—likely due to their use of fault-tolerance mechanisms. While these efforts contribute valuable taxonomies and analysis methods, they primarily focus on traditional web-scale services and do not address

the unique characteristics of failures in LLM-based applications.

Focusing specifically on LLM services, recent work has begun to systematically investigate their operational failure behaviors. Chu et al. [12] conduct an empirical characterization of outages and failure-recovery in public LLM services, analyzing failure recovery statistical properties, temporal patterns, co-occurrence, and the impact range of outage-causing incidents. Building on this foundation, Battaglini-Fischer et al. [8] introduce an open-source framework for collecting and analyzing incident reports across multiple LLM providers. Their platform facilitates structured data collection, incident analysis, and failure visualization. Complementing these system-level analyses, Yu et al. [42] present a comprehensive taxonomy of LLM system failures across six architectural layers, derived from a survey of 160 papers and repositories. While these studies provide valuable infrastructure and conceptual tools for LLM failure research, they are primarily built on data from official operator channels, and lack the complementary perspective offered by user-generated failure reports.

This gap—between operator-reported failure data and user-experienced service disruptions—remains insufficiently addressed. No prior work systematically integrates both operator and user reports to develop a comprehensive characterization of LLM service failures and recovery behaviors. This study seeks to fill that gap by combining heterogeneous data sources, enabling a more complete understanding of LLM service reliability through a multi-perspective analysis.

9 Conclusion

This study systematically investigates the reliability of LLM services by collecting and analyzing failure data from both operator and user perspectives. Unlike prior research that primarily relies on official incident reports, this work incorporates heterogeneous data sources, including official status pages and third-party user reporting platforms, to provide a more comprehensive and multi-perspective view of service failures.

A structured data pipeline was developed to collect and process failure reports into a unified data format. Based on the processed data, four analytical dimensions were explored: (1) failure-recovery modeling characterizes failure recovery patterns from both operator and user perspectives; (2) temporal analysis uncovers trends, as well as periodic and diurnal patterns in failure occurrences; (3) correlation analysis evaluates the extent to which user-reported volumes reflect the timing of officially acknowledged incidents and identifies failure co-occurrence patterns across services; and (4) consistency analysis quantifies the alignment between the two perspectives in terms of failure types and failure periods.

Based on these four types of analysis, this study summarizes a total of 28 important observations, which capture a broad spectrum of insights, including discrepancies in recovery durations and failure intervals, variations in temporal patterns, cross-service dependencies, and the degree of alignment between reporting perspectives. Together, these analyses and their corresponding findings address the research questions and highlight the importance of incorporating user-side signals into evaluations of LLM service reliability.

By systematically analyzing failure data from different sources, this study reveals both discrepancies and alignment patterns across LLM services and between reporting perspectives. It also contributes standardized tools for multi-source failure data collection and analysis, provides new empirical insights into LLM service failure behavior, and lays the groundwork for future work on predictive failure modeling and user-aware, real-time reliability monitoring systems.

During the entire research process, several non-trivial challenges were encountered. Among them, the most significant arose in the early stages of data collection and processing, which was the most time-consuming phase of the project. For instance, the diversity and frequent changes in web interfaces across platforms required the use of platform-specific scraping strategies tailored to different page structures. In the data processing stage, using LLMs to extract failure types from free text required extensive experimentation with prompt designs to identify the one that produced the most accurate results. Similarly, inferring user-reported failure periods involved testing different smoothing methods and tuning key parameters, such as the moving average window size, to precisely capture the periods during which users perceived service failures.

Overcoming these challenges significantly improved relevant technical skills. For example, web scraping skills were strengthened through the implementation of various platform-specific strategies. Additionally, practical experience was gained in building and managing large-scale data pipelines for both data collection and processing, enhancing the ability to design systems capable of handling real-world, noisy data. Finally, a deeper understanding of LLM services' operational behavior and reliability characteristics was developed, supporting future work involving the use and evaluation of LLMs.

Appendix

Structured Incident Report Extraction Protocol

We define a high-precision, semantically grounded protocol for extracting structured information from LLM service incident reports, obtained from official provider status pages. Each incident record is represented as a JSON object. The goal is to convert these records into a unified tabular dataset containing six key fields: `title`, `affected_components`, `impact_level`, `start_time`, `end_time`, and `failure_type`. All fields must be extracted via per-entry human-like reading and interpretation, without use of heuristic rules, templates, regular expressions, or keyword triggers.

- **Title**: Directly extracted from the `basic_info.title` field.
- **Affected Components**: Directly extracted from the `affected_components` field, and preserved as a list.
- **Impact Level**: Mapped from `basic_info.impact_level` using the following standardization:
  - `"none"`, `"minor"` → `Degraded performance`
  - `"major"` → `Partial outage`
  - `"critical"` → `Full outage`
- **Start and End Time**: These fields represent the actual time interval of the incident. The extraction process must follow a strict semantic-first principle:
  1. All `timeline.content` entries must be read to determine whether any natural language expression conveys a bounded time range (e.g., "from ... through ...", "between ... and ...", or equivalent).
  2. If such temporal expressions are present, the `start_time` and `end_time` must be extracted from them using semantic interpretation, with proper timezone normalization.
  3. Only if no content conveys a clear start and end time semantically may the timestamps be derived from the earliest and latest values in `timeline.timestamp`.

  Time normalization requirements:
  - All times must be converted to the UTC timezone and formatted as `YYYY-MM-DD HH:MM:SS`.
  - Timezones such as CST (UTC+8), PDT (UTC–7), and others must be correctly resolved based on textual context.
  - If the text specifies UTC explicitly (e.g., "18:00 UTC on April 8"), the resulting datetime object must be timezone-aware (i.e., not naive).
- **Failure Type**: This field denotes the underlying user-visible symptom category and must be assigned based solely on semantic understanding of the incident's `title` and `timeline.content`. The valid categories are:

- `Login` — user cannot log in or sign up
- `Inaccessible` — the service or specific features are unavailable
- `Slow` — latency or performance degradation
- `Error Received` — users encounter errors while interacting with the system
- `Others` — identifiable incidents that do not fit the above
- `Unknown` — no failure symptoms are clearly stated or inferable

The classifier must not rely on any form of keyword detection or pattern matching. Misclassification due to the presence of misleading terms (e.g., interpreting "Errors when logging in" as `Error Received` rather than `Login`) is unacceptable. The interpretation must be context-sensitive and human-level.

Failure of Methodological Consistency: Any extraction that skips the semantic judgment step, falls back to timestamp prematurely, uses keyword presence to drive classification, or produces inconsistent type decisions across semantically equivalent cases is considered invalid, even if the final values appear superficially correct.

References

[1] D. Anandayuvaraj, M. Campbell, A. Tewari, and J. C. Davis. FAIL: Analyzing Software Failures from the News Using LLMs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pages 506–518, New York, NY, USA, Oct. 2024. Association for Computing Machinery.

[2] Anthropic. Anthropic Partners with Google Cloud. `https://www.anthropic.com/news/anthropic-partners-with-google-cloud`, 2023.

[3] Anthropic. Incident: Elevated error rates in Claude.ai and Console. `https://status.anthropic.com/incidents/t39s5hjpbs9v`, 2024.

[4] Anthropic. Incident: Elevated errors for requests to the Anthropic API. `https://status.anthropic.com/incidents/j5cdq83mncqv`, 2024.

[5] Anthropic. Incident: Errors on Claude.ai and Console for logins and requests. `https://status.anthropic.com/incidents/cm416m2m0p83`, 2024.

[6] Anthropic. Incident: Outage affecting account creation and document upload. `https://status.anthropic.com/incidents/0dv5d8qb9gy2`, 2024.

[7] Anthropic. Release notes of Claude. `https://docs.anthropic.com/en/release-notes/claude-apps`, 2025.

[8] S. Battaglini-Fischer, N. Srinivasan, B. L. Szarvas, X. Chu, and A. Iosup. FAILS: A Framework for Automated Collection and Analysis of LLM Service Incidents. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering*, pages 187–194, May 2025. arXiv:2503.12185 [cs].

[9] Character.AI. Incident: Investigating an issue. `https://status.character.ai/incidents/s2ncdls90d80`, 2025.

[10] K. T. Chitty-Venkata, S. Raskar, B. Kale, F. Ferdaus, A. Tanikanti, K. Raffenetti, V. Taylor, M. Emani, and V. Vishwanath. LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators, Oct. 2024. arXiv:2411.00136 [cs].

[11] X. Chu, D. Hofstätter, S. Ilager, S. Talluri, D. Kampert, D. Podareanu, D. Duplyakin, I. Brandic, and A. Iosup. Generic and ML Workloads in an HPC Datacenter: Node Energy, Job Failures, and Node-Job Analysis. In *2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 710–719, Oct. 2024. ISSN: 2690-5965.

[12] X. Chu, S. Talluri, Q. Lu, and A. Iosup. An empirical characterization of outages and incidents in public services for large language models. In M. Litoiu, E. Smirni, A. V. Papadopoulos, and K. Wolter, editors, *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering, ICPE 2025, Toronto, ON, Canada, May 5-9, 2025*, pages 69–80. ACM, 2025.

[13] S. Cui, A. Patke, Z. Chen, A. Ranjan, H. Nguyen, P. Cao, S. Jha, B. Bode, G. Bauer, C. Narayanaswami, D. Sow, C. D. Martino, Z. T. Kalbarczyk, and R. K. Iyer. Characterizing GPU Resilience and Impact on AI/HPC Systems, Mar. 2025. arXiv:2503.11901 [cs].

[14] DeepSeek. GitHub repository of DeepSeek-R1. `https://github.com/deepseek-ai/DeepSeek-R1`, 2025.

[15] J. Duan, S. Zhang, Z. Wang, L. Jiang, W. Qu, Q. Hu, G. Wang, Q. Weng, H. Yan, X. Zhang, X. Qiu, D. Lin, Y. Wen, X. Jin, T. Zhang, and P. Sun. Efficient Training of Large Language Models on Distributed Infrastructures: A Survey, July 2024. arXiv:2407.20018 [cs].

[16] P. T. Endo, M. Rodrigues, G. E. Gonçalves, J. Kelner, D. H. Sadok, and C. Curescu. High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing*, 5(1):16, Oct. 2016.

[17] A. George, M. Wang, J. Hanley, G. W. Ransom, J. Bent, and C. Zimmer. From failure to insight: Analyzing disk breakdowns in large-scale hpc environments. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 484–495, 2024.

[18] A. Handler, K. R. Larsen, and R. Hackathorn. Large language models present new questions for decision support. *International Journal of Information Management*, 79:102811, Dec. 2024.

[19] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. D. Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799. PMLR, May 2019. ISSN: 2640-3498.

[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-Rank Adaptation of Large Language Models, Oct. 2021. arXiv:2106.09685 [cs].

[21] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo, Y. Wen, and T. Zhang. Characterization of Large Language Model Development in the Datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 709–729, Santa Clara, CA, Apr. 2024. USENIX Association.

[22] A. Kulkarni, Y. Zhang, J. R. A. Moniz, X. Ge, B.-H. Tseng, D. Piraviperumal, S. Swayamdipta, and H. Yu. Evaluating Evaluation Metrics – The Mirage of Hallucination Detection, Apr. 2025. arXiv:2504.18114 [cs].

[23] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[24] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari. LLM Inference Serving: Survey of Recent Advances and Opportunities, July 2024. arXiv:2407.12391 [cs].

[25] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh. Reliability and high availability in cloud computing environments: a reference roadmap. *Human-centric Computing and Information Sciences*, 8(1):20, July 2018.

[26] S. Milová. Failure Modes of Large Language Models. June 2023. Accepted: 2023-07-24T22:55:05Z Publisher: Univerzita Karlova, Fakulta sociálních věd.

[27] NerdyNav. 107+ ChatGPT Statistics and User Numbers (May 2025). `https://nerdynav.com/chatgpt-statistics/?utm_source=chatgpt.com`, 2025.

[28] OpenAI. Elevated API and ChatGPT Latency and Error Rate. `https://status.openai.com/incidents/01JMYB51KQVDVWS4HVVSN6MK99`, 2024.

[29] OpenAI. Chatgpt overview. `https://openai.com/chatgpt/overview/`, 2025.

[30] OpenAI. Chatgpt search is seeing elevated failures and high latency. `https://status.openai.com/incidents/01JVF6PJN0JWJYNGS06BVYSH7Z`, 2025.

[31] OpenAI. Incident: Increased errors for ChatGPT. `https://status.openai.com/incidents/01JMYB40J9DVR9D13RJN1W5SWV`, 2025.

[32] Premier Cloud. Simplify complex tasks with Gemini on Google Cloud. `https://premiercloud.com/gemini-google-cloud/`, 2025.

[33] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23, pages 491–514, New York, NY, USA, Mar. 2023. Association for Computing Machinery.

[34] S. Talluri, D. Niewenhuis, X. Chu, J. Kyselica, M. Cetin, A. Balgavy, and A. Iosup. Cloud Uptime Archive: Open-Access Availability Data of Web, Cloud, and Gaming Services, Apr. 2025. arXiv:2504.09476 [cs].

[35] J. Tie, B. Yao, T. Li, S. I. Ahmed, D. Wang, and S. Zhou. LLMs are Imperfect, Then What? An Empirical Study on LLM Failures in Software Engineering, Nov. 2024. arXiv:2411.09916 [cs].

[36] P. Vellaisamy, T. Labonte, S. Chakraborty, M. Turner, S. Sury, and J. P. Shen. Characterizing and Optimizing LLM Inference Workloads on CPU-GPU Coupled Architectures, Apr. 2025. arXiv:2504.11750 [cs].

[37] J. Vendrow, E. Vendrow, S. Beery, and A. Madry. Do Large Language Model Benchmarks Test Reliability?, Feb. 2025.

[38] Y. Wang, Y. Chen, Z. Li, Z. Tang, R. Guo, X. Wang, Q. Wang, A. C. Zhou, and X. Chu. Towards efficient and reliable llm serving: A real-world workload study. *arXiv e-prints*, pages arXiv–2401, 2024.

[39] K. Wiggers. ChatGPT isn't the only chatbot that's gaining users. https://techcrunch.com/2025/04/01/chatgpt-isnt-the-only-chatbot-thats-gaining-users/, 2025.

[40] Wikipedia. ChatGPT. https://en.wikipedia.org/wiki/ChatGPT, 2025.

[41] Y. Xiang, X. Li, K. Qian, W. Yu, E. Zhai, and X. Jin. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production, May 2025. arXiv:2505.09999 [cs].

[42] G. Yu, G. Tan, H. Huang, Z. Zhang, P. Chen, R. Natella, and Z. Zheng. A Survey on Failure Analysis and Fault Injection in AI Systems, June 2024. arXiv:2407.00125 [cs].