09

最近発生した 3 件の問題のポストモーテム

— A postmortem of three recent issues —

公開日 2025-09-17
原題 A postmortem of three recent issues
著者 Anthropic Engineering Team
原文 https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues
翻訳 Claude（機械翻訳／Anthropic）
編集 2026-04-09

最近発生した 3 件の問題のポストモーテム

8 月から 9 月上旬にかけて、3 つのインフラストラクチャ上のバグが断続的に Claude の応答品質を低下させました。現在これらの問題は解決済みであり、何が起きたかを説明したいと思います。

8 月上旬、一部のユーザーが Claude の応答品質の劣化を報告し始めました。初期の報告は、ユーザーフィードバックの通常のばらつきと区別するのが難しいものでした。しかし 8 月下旬、報告の頻度と持続性の増加を受けて調査を開始し、3 つの別々のインフラストラクチャバグを発見するに至りました。

はっきり述べます: **需要、時間帯、サーバー負荷を理由にモデル品質を下げることは決してありません。**ユーザーが報告した問題は、すべてインフラバグに起因するものでした。

ユーザーが Claude に一貫した品質を期待していることは認識しており、インフラ変更がモデル出力に影響しないよう非常に高い基準を維持しています。今回の一連のインシデントでは、その基準を満たせませんでした。以下のポストモーテムでは、何が悪かったのか、検知と解決に想定以上の時間を要した理由、そして同様のインシデントを防ぐために変更している点を説明します。

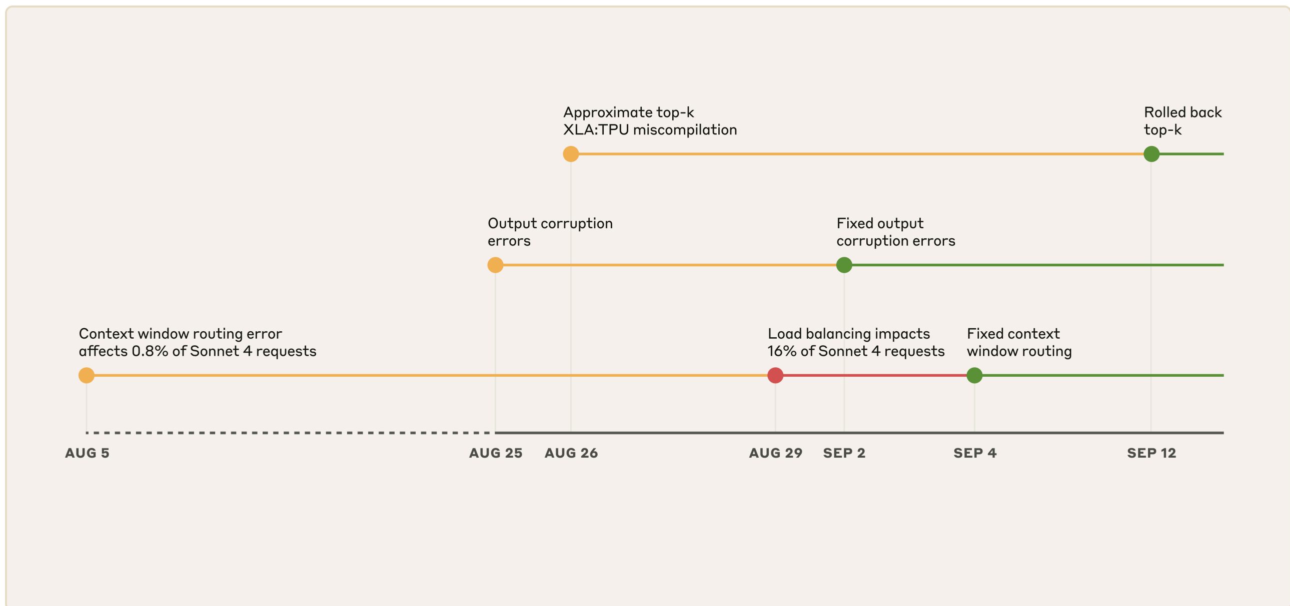
通常、私たちはインフラについてこれほどの技術詳細を共有しませんが、今回の問題の範囲と複雑性を鑑みて、より包括的な説明が必要だと判断しました。

Claude をどうスケールさせて提供しているか

私たちは Claude を、自社 API、Amazon Bedrock、Google Cloud の Vertex AI を通じて数百万のユーザーに提供しています。Claude は、AWS Trainium、NVIDIA GPU、Google TPU という複数のハードウェアプラットフォームにデプロイされています。このアプローチは、世界中のユーザーにサービスを提供するのに必要な容量と地理的分散を提供します。

各ハードウェアプラットフォームは異なる特性を持ち、それぞれ固有の最適化を必要とします。そうした違いがあっても、モデル実装の同等性には厳しい基準を設けています。私たちの狙いは、どのプラットフォームがリクエストを処理してもユーザーが同じ品質の応答を得られることです。この複雑さは、インフラ変更がすべてのプラットフォームと構成にわたる注意深い検証を必要とすることを意味します。

事象のタイムライン



Claude API 上の事象の例示的タイムライン。黄: 問題検知、赤: 劣化悪化、緑: 修正デプロイ。

これらのバグが重なり合っていたため、診断は特に困難でした。最初のバグは 8 月 5 日に導入され、Sonnet 4 への約 0.8% のリクエストに影響しました。さらに 2 つのバグが 8 月 25 日と 26 日のデプロイから生まれました。

初期の影響は限定的でしたが、8 月 29 日のロードバランシング変更で影響を受けるトラフィックが増え始めました。これにより、正常な性能を見続けるユーザーがいる一方で、より多くのユーザーが問題を経験し、混乱し矛盾する報告が生まれました。

3 つの重なり合う問題

以下に、劣化を引き起こした 3 つのバグ、それが起きた時期、そして解決方法を記します。

1. コンテキストウィンドウのルーティングエラー

8 月 5 日、一部の Sonnet 4 リクエストが、近日公開予定の 1M トークンコンテキストウィンドウ向けに構成されたサーバーに誤ルーティングされていました。このバグは当初 0.8% のリクエストに影響しました。8 月 29 日、ルーチンのロードバランシング変更が、意図せず短コンテキストリクエストの 1M コンテキストサーバーへのルーティング数を増やしました。最悪の時間帯だった 8 月 31 日には、Sonnet 4 リクエストの 16% が影響を受けました。

この期間中にリクエストした Claude Code ユーザーの約 30% が、少なくとも 1 メッセージを誤ったタイプのサーバーにルーティングされ、劣化した応答を受けました。Amazon Bedrock では、誤ルーティングされたトラフィックは 8 月 12 日からの全 Sonnet 4 リクエストの 0.18% でピークに達しました。Google Cloud の Vertex AI では、8 月 27 日から 9 月 16 日の間で 0.0004% 未満のリクエストが影響を受けました。

ただし、私たちのルーティングは「スティッキー（sticky）」であるため、一部のユーザーはより深刻な影響を受けました。つまり、いったん誤ったサーバーでリクエストが処理されると、後続のフォローアップも同じ誤ったサーバーで処理される可能性が高かったのです。

解決: 短コンテキストと長コンテキストのリクエストが正しいサーバープールに振り分けられるようルーティングロジックを修正しました。修正は 9 月 4 日にデプロイしました。自社プラットフォームと Google Cloud Vertex AI へのロールアウトは 9 月 16 日までに、AWS Bedrock へは 9 月 18 日までに完了しました。

2. 出力の破損

8 月 25 日、Claude API の TPU サーバーに誤った構成をデプロイし、トークン生成中にエラーが発生するようにしてしまいました。実行時性能最適化によって引き起こされた問題で、文脈を考えるとめったに生成されるべきでないトークンに時折高い確率が割り当てられました――たとえば英語のプロンプトへの応答中にタイ語や中国語の文字が生成されたり、コードに明らかな構文エラーが出たりしました。英語で質問した一部のユーザーは、たとえば応答の途中に「สวัสดี」を見たかもしれません。

この破損は 8 月 25～28 日の Opus 4.1 と Opus 4 への、および 8 月 25 日～9 月 2 日の Sonnet 4 へのリクエストに影響しました。サードパーティプラットフォームはこの問題の影響を受けませんでした。

解決: 問題を特定し、9 月 2 日に変更をロールバックしました。デプロイプロセスに、予期しない文字出力を検知するテストを追加しました。

3. 近似 top-k の XLA:TPU ミスコンパイル

8 月 25 日、テキスト生成中の Claude のトークン選択の仕方を改善するコードをデプロイしました。この変更が図らずも XLA:TPU[1] コンパイラの潜在バグをトリガーし、Claude Haiku 3.5 へのリクエストに影響を及ぼしたことが確認されています。

Claude API 上の Sonnet 4 と Opus 3 の一部にも影響した可能性があると考えています。サードパーティプラットフォームは影響を受けていません。

解決: まず Haiku 3.5 への影響を観測し、9 月 4 日にロールバックしました。その後、Opus 3 の問題についてこのバグと整合するユーザー報告が見られ、9 月 12 日にロールバックしました。広範な調査の末、Sonnet 4 でこのバグを再現できなかったものの、念のため Sonnet 4 もロールバックしました。

同時に、(a) XLA:TPU チームと協力してコンパイラバグの修正に取り組んでおり、(b) 精度を高めた厳密 top-k を使う修正をロールアウトしました。詳細は以下の深掘りを参照してください。

XLA コンパイラバグの詳細

これらの問題の複雑さを示すために、XLA コンパイラバグがどう顕在化し、なぜ診断が特に困難だったかをここで解説します。

Claude がテキストを生成するとき、取り得る各次単語の確率を計算し、その確率分布から無作為にサンプリングします。意味不明な出力を避けるために「top-p サンプリング」を使い、累積確率がしきい値(通常 0.99 または 0.999)に達する単語だけを考慮します。TPU 上では、私たちのモデルは複数のチップにまたがって動き、確率計算は異なる場所で行われます。これらの確率をソートするには、チップ間でデータを協調させる必要があり、複雑です[2]。

2024 年 12 月、私たちは TPU 実装が温度(temperature) がゼロのときに時折最も確率の高いトークンを落とすことを発見しました。このケースを修正するワークアラウンドをデプロイしました。

```
2679 +        # This is a hack for top_p=0 aka temperature=0.0. Due to odd jax issues, it's
2680 +        # possible that (probs >= probs.max(-1)) is all False. So we manually always
2681 +        # preserve the top-logit in each row.
2682 +        argmax_mask = jnp.arange(probs.shape[-1])[None, None] == probs.argmax(
2683 +            -1, keepdims=True
2684 +        )
2685 +        logits = jnp.where((probs >= prob_limit) | argmax_mask, logits, logits - 50000)
```

temperature = 0 で予期せぬトークン落としバグを回避するための 2024 年 12 月パッチのコード。

根本原因は混合精度演算でした。私たちのモデルは次トークン確率を bf16(16 ビット浮動小数点)で計算します。しかしベクトルプロセッサは fp32 ネイティブ であるため、TPU コンパイラ(XLA)は一部の操作を fp32 (32 ビット)に変換して実行時を最適化できます。この最適化パスは `xla_allow_excess_precision` フラグで制御され、デフォルトは true です。

これがミスマッチを生みました——最も高確率のトークンに合意すべき操作が、異なる精度レベルで走っていたのです。精度の不一致により、どのトークンが最高確率かが一致せず、最高確率トークンが完全に候補から消えることがありました。

8 月 26 日、精度問題を修正し、top-p しきい値に達する限界付近の確率の扱いを改善するために、サンプリングコードを書き直したものをデプロイしました。しかし、これらの問題を修正したことで、より厄介な問題が露出しました。

めたもの。実際には `xla_allow_excess_precision` フラグの期待どおりの振る舞いだった。

```diff
+ """Minimal reproduction of JAX optimization 'bug' with bf16 conversions.
+
+ This test demonstrates a JAX footgun where comparing bf16 values
+ after type conversions produces incorrect results without an optimization barrier.
+
+ Issue goes away with `XLA_FLAGS=--xla_allow_excess_precision=False`.
+
+ Must be run on TPUs to reproduce the issue.
+ Also see test_mask_top_k_and_top_p_logits_jax_bug.
+ """
+
+ import tempfile
+
+ import jax
+ import jax.numpy as jnp
+
+
+ def test_jax_bf16_optimization_bug():
+     """Minimal test case for JAX bf16 optimization 'bug'."""
+
+     key = jax.random.PRNGKey(42)
+     x = jax.random.normal(key, (2, 1, 1024), dtype=jnp.float32)
+
+     def f(inp: jax.Array, workaround: bool) -> jax.Array:
+         x = jax.nn.softmax(inp, axis=-1)
+         x = x.astype(jnp.bfloat16)
+
+         y = jnp.max(x, axis=-1, keepdims=True)
+
+         if workaround:
+             x, y = jax.lax.optimization_barrier((x, y))
+
+         x = x.astype(jnp.float32)
+         y = y.astype(jnp.float32)
+
+         return jnp.where(x >= y, inp, -50000)
+
+     jitted_f = jax.jit(f, static_argnames=["workaround"])
+     result_without_barrier = jitted_f(x, workaround=False)
+     result_with_barrier = jitted_f(x, workaround=True)
+
+     # Count how many values were preserved (should be 2 total)
+     preserved_without = jnp.sum(result_without_barrier > -50000)
+     preserved_with = jnp.sum(result_with_barrier > -50000)
+
+     print(f"\nResults:")
+     print(f"  Without barrier: {preserved_without} values preserved (expected: 2)")
+     print(f"  With barrier:    {preserved_with} values preserved (expected: 2)")
```

8月11日の変更にマージされた最小再現コード。2024年12月に回避していた「バグ」の根本原因を突き止めたもの。実際には `xla_allow_excess_precision` フラグの期待どおりの振る舞いだった。

修正では 12 月のワークアラウンドも取り除きました。根本原因を解決したと信じていたからです。これが、近似 top-k 操作に潜むより深いバグに繋がりました——近似 top-k は最高確率トークンを素早く見つけるための性能最適化です[3]。この近似が、特定のバッチサイズやモデル構成のときに限って、完全に誤った結果を返すことがありました。12 月のワークアラウンドは、不用意にこの問題を覆い隠していたのです。

```
import jax.lax as lax
import jax.numpy as jnp
import numpy as np

k = 256
N = 12000

for i in range(N):
    arr = jnp.zeros((N,))
    arr = arr.at[i].set(1.0)

    # Run approx_max_k
    approx_values, approx_indices = lax.approx_max_k(arr, k=k)

    # Run top_k (exact)
    exact_values, exact_indices = lax.top_k(arr, k=k)

    if approx_values[0] != exact_values[0]:
        print(f"Diff at {N=} {i=} topk={exact_values[0]} approxk={approx_values[0]}")
```

basically just creates an array of all zeroes but with a single one.
on our side running on v5e i see that for i>=10240 the approx version always fails to find the max value.

近似 top-k バグの再現コード。アルゴリズムを開発した XLA:TPU エンジニアに共有した Slack メッセージ。CPU で走らせれば正しい結果を返す。

バグの振る舞いはもどかしいほど一貫しませんでした。直前や直後に走る操作、デバッグツールの有効化の有無といった無関係な要因で振る舞いが変わるのです。同じプロンプトが、あるリクエストでは完璧に動き、次のリクエストでは失敗することもありました。

調査中、厳密 top-k 操作がかつてのような禁止的な性能ペナルティをもはや持たないことも発見しました。近似から厳密 top-k に切り替え、追加の操作を fp32 精度で標準化しました[4]。モデル品質は交渉の余地がないので、わずかな効率への影響は受け入れました。

検知が難しかった理由

私たちの検証プロセスは通常、ベンチマークに加えて安全性評価と性能指標に依拠します。エンジニアリングチームはスポットチェックを行い、まず小規模な「カナリア」グループにデプロイします。

今回の問題は、私たちがもっと早く特定すべきだった重大な穴を露呈しました。走らせていた evaluation は、ユーザーが報告していた劣化を捉えられなかったのです。Claude は単発のミスからうまく回復することが多いことも一因です。自分たち自身のプライバシー慣行も調査を難しくしました。内部のプライバシー／セキュリティ管理は、特にユーザーがフィードバックとして報告していないやり取りについて、エンジニアがユーザーとの Claude のやり取りにアクセスできる方法と時期を制限します。これはユーザーのプライバシーを守る一方で、バグを特定・再現するために必要な問題のあるやり取りをエンジニアが検査するのを妨げます。

各バグは、異なるプラットフォームで異なる頻度で異なる症状を生みました。これが、単一原因を指し示さない混乱した報告の集合を作り出しました。ランダムで一貫性のない劣化に見えたのです。

より根本的には、私たちはノイズの多い evaluation に頼りすぎていました。オンラインで報告が増えていることは認識していましたが、最近の変更との関連を明確にする方法が欠けていました。8 月 29 日に否定的な報告が急増したとき、標準的なロードバランシング変更との関連に即座には気づかなかったのです。

私たちが変えていること

インフラを改善し続けると同時に、Claude を提供するすべてのプラットフォームで、上記のようなバグを評価・予防する方法も改善しています。以下が変えていることです。

- **より感度の高い evaluation**: 任意の問題の根本原因を見つけるために、動作する実装と壊れた実装をより確実に見分けられる evaluation を開発しました。これらを改善し続け、モデル品質をより近くで監視します。
- **より多くの場所で品質 evaluation を**: 普段から定期 evaluation を走らせていますが、コンテキストウィンドウロードバランシングエラーのような問題を捕まえるために、実際の本番システム上で連続的に走らせます。
- **より高速なデバッグツール**: ユーザープライバシーを犠牲にすることなく、コミュニティ由来のフィードバックをよりよくデバッグするインフラとツールを開発します。加えて、ここで開発した専用ツールのいくつかは、将来同様のインシデントが発生したときの回復時間短縮に使います。

eval と監視は重要です。しかし今回のインシデントは、Claude の応答が通常の基準に達していないときに、ユーザーからの継続的なシグナルも必要であることを示しました。観察された具体的な変化の報告、遭遇した予期せぬ振る舞いの例、ユースケース横断のパターンが、問題の切り分けに役立ちました。

ユーザーの皆さんが引き続き直接フィードバックを送ってくれることは特に役立ちます。Claude Code の `/bug` コマンドや、Claude アプリの「いいえ」ボタンを使ってください。開発者や研究者はしばしば、私たちの内部テストを補完するモデル品質評価の新しく興味深い方法を作り出します。共有したい場合は feedback@anthropic.com までご連絡ください。

これらの貢献をしてくれるコミュニティに改めて感謝します。

謝辞

執筆は Sam McAllister。Stuart Ritchie、Jonathan Gray、Kashyap Murali、Brennan Saeta、Oliver Rausch、Alex Palcuie そしてその他多数に感謝します。

---

[1] XLA:TPU は、XLA High Level Optimizing 言語——しばしば JAX で書かれる——を TPU 機械命令に変換する最適化コンパイラ。

[2] 私たちのモデルは単一チップには大きすぎるため、数十のチップ以上に分割されており、ソート操作は分散ソートになる。TPU（GPU や Trainium と同様）は CPU とも性能特性が異なり、逐次アルゴリズムではなくベクトル化された操作を使う異なる実装手法を要する。

[3] 近似操作を使っていたのは、大幅な性能改善が得られたからである。近似は、最低確率トークンでの潜在的な不正確さを受け入れることで動くが、通常は品質には影響しないはずである——バグが最高確率トークンを落とさせてしまった場合を除いて。

[4] 正しくなった top-k 実装は、top-p しきい値近辺のトークン包含にわずかな差をもたらすことがあり、まれにユーザーが top-p の選択を再チューニングすることで恩恵を受けるかもしれない。