22

エージェント型コーディング eval におけるインフラノイズの定量化

— Quantifying infrastructure noise in agentic coding evals —

原題	Quantifying infrastructure noise in agentic coding evals
著者	Anthropic Engineering Team
原文	https://www.anthropic.com/engineering/infrastructure-noise
翻訳	Claude（機械翻訳／Anthropic）
編集	2026-04-09

エージェント型コーディング eval におけるインフラノイズの定量化

SWE-bench や Terminal-Bench のようなエージェント型コーディングのベンチマークは、フロンティアモデルのソフトウェアエンジニアリング能力を比較するために広く使われており、リーダーボードの上位はわずか数パーセントポイントで分かれていることもしばしばです。これらのスコアは、モデル間の相対的な能力差を表す精密な計測値として扱われ、どのモデルをデプロイするかという意思決定にもますます影響を与えるようになっています。しかし私たちは、インフラ構成の違いだけでこの差を上回るほどの差分が生まれうることを発見しました。社内実験では、Terminal-Bench 2.0 において、最もリソースが潤沢な構成と最も乏しい構成との差は 6 パーセントポイント（$p < 0.01$）にのぼりました。

静的なベンチマークはモデルの出力を直接採点するもので、ランタイム環境は結果に影響しません。エージェント型コーディングの eval はそれとは異なります。モデルには、プログラムを書き、テストを実行し、依存関係をインストールし、複数ターンにわたって反復作業を行うための完全な環境が与えられます。ランタイムはもはや受動的なコンテナではなく、問題解決プロセスに不可欠な構成要素となります。リソース予算や時間制限が異なる 2 つのエージェントは、もはや同じテストを受けているとは言えないのです。

eval の開発者たちはこの点を考慮し始めています。たとえば Terminal-Bench 2.0 では、最新の 2.0 リリースにおいてタスクごとに推奨 CPU と RAM が指定されています。しかし、リソースを指定することと、それを一貫して強制することは同じではありません。さらに、強制方法そのものがベンチマークが実際に測定しているものを変えてしまうことを私たちは発見しました。

そこに至った経緯

私たちは Terminal-Bench 2.0 を Google Kubernetes Engine クラスタ上で実行しています。セットアップを調整している最中、自分たちのスコアがベンチマークの公式リーダーボードと一致しないこと、そしてインフラのエラー率が驚くほど高いことに気づきました。タスクの最大 6% が pod のエラーによって失敗しており、その大半はモデルがタスクを解く能力とは無関係なものでした。

スコアの食い違いは、リソース強制の方法に起因していました。私たちの Kubernetes 実装は、タスクごとのリソース指定を「下限」と「絶対上限」の両方として扱っていたのです。各コンテナには指定されたリソースが保証される一方で、それを超過した瞬間に kill されるという仕組みでした。コンテナランタイムは 2 つの別々のパラメータでリソースを強制します。1 つは事前に予約される「保証された割り当て」、もう 1 つはそれを超えるとコンテナが kill される「ハードリミット」です。これらが同じ値に設定されると、一時的なスパイクのための余裕がゼロになります。一瞬のメモリ変動が、本来であれば成功していたコンテナを OOM kill してしまうのです。こ

れに対処するため、Terminal-Bench のリーダーボードは別のサンドボックスプロバイダを使っており、その実装はより寛容で、インフラの安定性を優先するために一時的なオーバーアロケーションを許容し、コンテナを終了させません。

この発見は、より大きな問いを投げかけました。リソース構成は eval スコアにどれだけ影響するのでしょうか?

足場（scaffold）の影響を定量化するため、私たちは 6 つのリソース構成にわたって Terminal-Bench 2.0 を実行しました。タスクごとの仕様を厳密に強制する設定（1x、下限と上限の両方として機能する）から、完全に上限なしの設定までです。それ以外はすべて固定しました。同じ Claude モデル、同じハーネス、同じタスクセットです。

実験の結果、リソースの余裕が増えるにつれて成功率は上昇しました。これは主にインフラエラー率が各段階で単調に減少したことによるもので、厳密強制では 5.8% だったものが、上限なしでは 0.5% にまで下がりました。厳密強制から 3x の余裕までの低下（5.8% から 2.1%）は $p < 0.001$ で有意でした。余裕が増えれば、割り当て超過によって kill されるコンテナは減ります。

1x から 3x の範囲では、成功スコアはノイズの範囲内で変動しました（$p=0.40$）。1x でクラッシュしていたタスクの大半は、いずれにせよ失敗していたであろうものでした——実際、データ上もそれが観測されました。エージェントは探索を行い、リソースの壁にぶつかってプリエンプトされますが、そもそも正解への道筋に乗っていなかったのです。

しかし 3x あたりからこの傾向は変わります。成功率がインフラエラーの減少よりも速く上昇し始めるのです。

3x から上限なしまでの間に、インフラエラーはさらに 1.6 パーセントポイント減少する一方、成功率はほぼ 4 パーセントポイント跳ね上がります。追加リソースは、潤沢な割り当てがあって初めて機能するアプローチをエージェントが試せるようにします。たとえば、大きな依存関係を引き込んだり、コストの高いサブプロセスを起動したり、メモリ集約的なテストスイートを走らせたりといったものです。リソース上限なしでは、1x に対する合計上昇幅は +6 パーセントポイント（$p < 0.01$）となります。マージンの部分では、`rstan-to-pystan` や `compile-compcert` のようなタスクが、メモリの余裕を得ることで成功率を大きく改善しました。

これが計測にどう影響するか

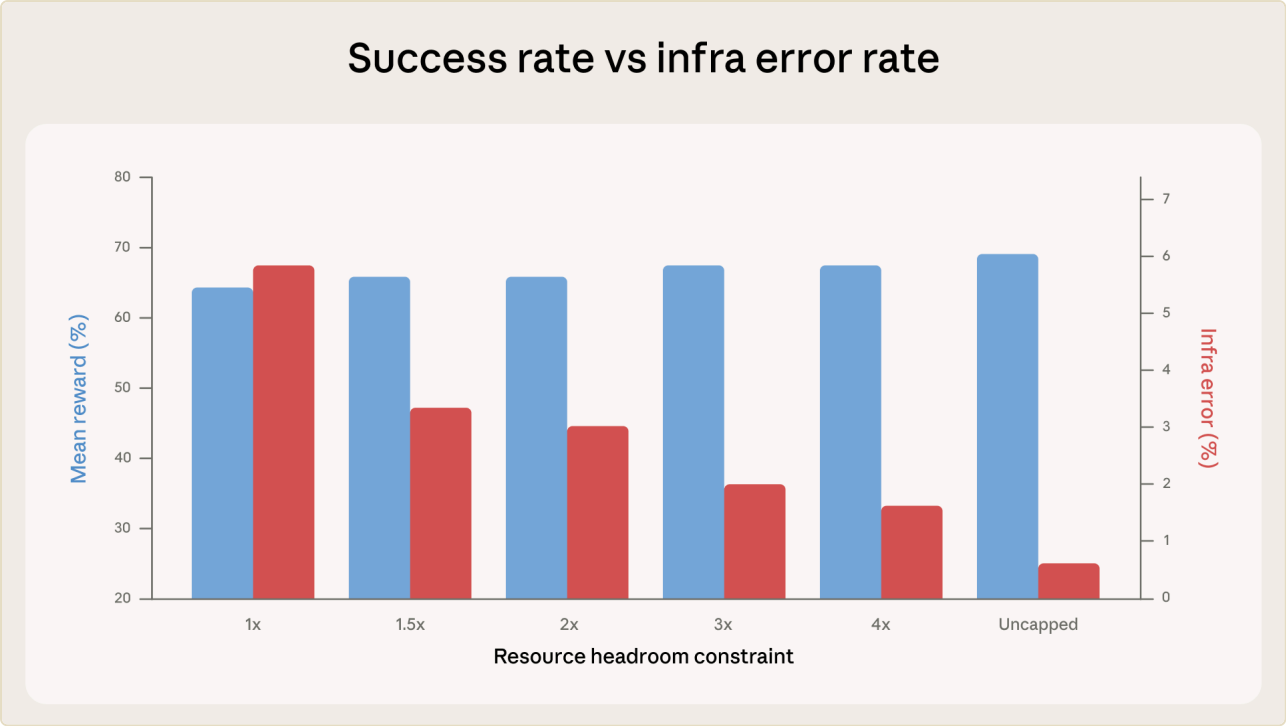
Terminal-Bench の仕様のおおむね 3x までの範囲では、追加リソースはインフラの信頼性問題、すなわち一時的なリソーススパイクを解消する役割を果たします。Terminal-Bench メンテナが使っているサンドボックスプロバイダは、これを暗黙のうちに裏で行っているのです。eval は安定性を増す一方で、難度が下がるわけではありません。

しかし 3x を超えると、追加リソースは、エージェントがそれまで解けなかった問題を能動的に解けるように手助けし始めます。これは、リソース制限が eval の測定対象そのものを変えうることを示しています。厳しい制限は、知らず知らずのうちに非常に効率的な戦略を有利にし、潤沢な制限は寛容で、利用可能なリソースをよりうまく活用できるエージェントを有利にします。

無駄のない効率的なコードを高速に書くエージェントは、厳しい制約下で良い成績を出すでしょう。重量級ツールでブルートフォース的に解くエージェントは、寛容な制約下で良い成績を出すでしょう。どちらをテストするのも正当ですが、リソース構成を明示せずに 1 つのスコアにまとめてしまうと、両者の違いや実世界での汎化可能性が読み取りにくくなります。

ベイジアンネットワークのフィッティングを必要とする Terminal-Bench のタスク `bn-fit-modify` では、いくつかのモデルは最初の一手として標準的な Python データサイエンススタックである `pandas`、`networkx`、`scikit-learn`、そしてそれらのツールチェーン一式をインストールします。潤沢な制限下ではこれが機能します。しかし厳しい制限下では、エージェントが解法コードを 1 行も書かないうちに、インストール中に pod のメモリが尽きてしまいます。よりリーンな戦略（標準ライブラリだけで数式をゼロから実装する）も存在し、それをデフォルトで採用するモデルもあります。一方でそうしないモデルもあります。モデルによってデフォルトのアプローチは異なり、リソース構成によってそのうちのどのアプローチが成功するかが決まるのです。私たちはこの中心的な発見を、複数の Anthropic モデルにわたって再現しました。効果の方向は一貫していましたが、その大きさは異なりました。同じ傾向は Claude 以外のモデルにもあてはまるように見えますが、厳密にはテストしていません。

このパターンが Terminal-Bench 以外の eval にもあてはまるかを確かめるため、SWE-bench でクロスオーバー実験も行いました。227 問題、それぞれ 10 サンプルずつで、利用可能な総 RAM をベースラインの最大 5x まで変動させました。同じ効果が見られましたが、その大きさはより小さいものでした。スコアは RAM とともに単調に増加しましたが、5x での値は 1x よりわずか 1.54 パーセントポイント高いだけでした。SWE-bench のタスクはリソース集約度がより低いため、効果が小さいのは想定どおりですが、それでもリソース割り当てが中立ではないことを示しています。



その他のばらつきの源

リソース割り当てだけが隠れた変数というわけではありません。特定の構成では、時間制限もまた役割を果たし始めます。

原則として、評価セットアップのあらゆる要素が最終スコアに影響を与えうるのです。クラスタの健全性からハードウェアの仕様、並列度、さらには egress 帯域に至るまで。エージェント型 eval は構造上エンドツーエンドのシステムテストであり、そのシステムのどの構成要素も交絡因子になりえます。たとえば私たちは、合格率が時間帯によって変動するという事例的な観測も得ています。これはおそらく API のレイテンシがトラフィックパターンやインシデントによって変わるためです。この効果を正式に定量化したわけではありませんが、より大きな点を示しています。「モデルの能力」と「インフラの挙動」の境界は、単一のベンチマークスコアが示唆するよりもはるかに曖昧です。モデル提供者は専用ハードウェアを割り当てることで自社の eval インフラをこれから守ることができますが、外部の評価者には同じことを容易に行うことはできません。

公開ベンチマークは通常、純粋なモデル能力を測ることを目的としていますが、実際にはインフラの癖と能力を混同してしまうリスクがあります。スタック全体のエンドツーエンドテストが可能になるという点ではこれが望ましい場合もありますが、多くの場合はそうではありません。公開を前提としたコーディング eval については、複数の時刻、複数の日にわたって実行することでノイズを平均化できるでしょう。

私たちの推奨

理想的なシナリオは、各 eval をまったく同じハードウェア条件下で実行することです——eval を走らせる足場と推論スタックの両方を含めて——そうすれば全面的な完全な再現性が保証されます。しかし、これは必ずしも実用的ではないでしょう。

コンテナランタイムが実際にリソースをどう強制するか——保証された割り当てと、それとは別のハードキル閾値という 2 つの値で——を踏まえると、eval はタスクごとにこれら両方のパラメータを指定すべきであり、単一の固定値ではないことを推奨します。単一の正確な仕様は、保証された割り当てとキル閾値を等しくしてしまい、余裕がゼロになります。私たちが 1x で記録した一時的なメモリスパイクは、それだけで eval を不安定にするのに十分です。2 つのパラメータを分離することで、コンテナに無用な OOM kill を避けるだけの呼吸の余地を与えつつ、スコアの水増しを防ぐハードな上限を強制することができます。

両者の幅は、下限と上限でのスコアが互いにノイズの範囲内に収まるようにキャリブレーションされるべきです。たとえば Terminal-Bench 2.0 では、タスクごとの仕様に対する 3x の上限が、インフラエラー率をおよそ 3 分の 1（5.8% から 2.1%、$p < 0.001$）に削減する一方で、スコアの上昇は控えめでノイズの範囲内に十分収まりました（$p = 0.40$）。これは妥当なトレードオフです。意味のあるリソース圧力を取り除くことなく、インフラ由来の交絡因子をほぼ中立化できているのです。正確な倍率はベンチマークやタスクの分布によって異なるため、その値は報告すべきですが、経験的にキャリブレーションするという原則は一般的に通用します。

なぜこれが重要か

これらの発見には、eval インフラを超えた実務的な帰結があります。ベンチマークスコアは意思決定の入力としてますます使われていますが、その注目度（と依存度）の高まりに、実行や報告における厳密さが必ずしも追いついてきたわけではありません。現状では、リーダーボード上の 2 ポイントの差は本物の能力差を反映しているかもしれませんし、あるいは一方の eval がより強力なハードウェアで、または運の良い時刻に走った結果かもしれません。あるいはその両方かもしれません。セットアップ構成が公開（または標準化）されていない限り、関心を持つ第三者がわざわざ同一条件下で結果を再現する手間をかけない限り、外部からは判断が困難です。

Anthropic のような研究所にとっての含意は、エージェント型 eval のリソース構成は一級の実験変数として扱うべきであり、プロンプト形式やサンプリング温度と同じ厳密さで文書化・コントロールされるべきだということです。ベンチマークのメンテナにとっては、（Terminal-Bench 2.0 がそうしているように）推奨リソース仕様を公表するだけでも大きな前進であり、強制方法まで明示すれば私たちが指摘したギャップを埋められるでしょう。そしてベンチマーク結果を消費するすべての人にとっての中心的な教訓は、エージェント型 eval における小さなスコア差は、報告されている数字の精度が示唆するよりも大きな不確実性を伴うということです——特に、いくつかの交絡因子はそもそもコントロールするのが非常に難しいからです。

リソースの方法論が標準化されるまで、私たちのデータは、リーダーボード上の 3 パーセントポイント未満の差は、eval 構成が文書化・一致させられるまで懐疑的に見るべきであることを示唆しています。Terminal-Bench における中程度のリソース構成範囲で観測されたばらつきは、わずか 2 パーセントポイント弱です。素朴な二項分布の信頼区間はすでに 1〜2 パーセントポイントに及びます。私たちがここで記録したインフラの交絡因子は、その内側ではなく、その上に積み重なります。割り当て範囲の両極端では、ばらつきは 6 にまで達します。

数ポイントのリードは本物の能力差を示しているかもしれません——あるいは単に、より大きな VM のことかもしれないのです。

謝辞

執筆は Gian Segato。貢献に対し Nicholas Carlini、Jeremy Hadfield、Mike Merrill、Alex Shaw に特別に感謝します。本稿はコーディングエージェントの評価に取り組む複数チームの集合的な努力を反映しています。貢献に興味のある候補者の方は、ぜひ anthropic.com/careers からご応募ください。