検証ハーネス — AIが生成した実装を実測で確かめる

2026年11月14日
SimpleModeling

AIが生成したプログラムは、出力が正しくても、余分な処理で性能目標を外れることがあります。前編の観測モデルは、実行条件、出力、呼び出しの経路や処理時間を、同じ実行の記録として結び付けます。検証ハーネスは、この記録を要求や設計から定めた適合条件に照らし、変更を採用できるか判断する枠組みです。

1　この記事の構成

前半は、検証ハーネスの適合条件と、判断の根拠を得るための標準機能を説明します。中盤の図1では、実運用の観測とCorpusによる再検証の関係を示します。後半は、注文書処理を題材に、コードの修正と、プログラムが実行時に呼び出すAIモデルの構成変更について、何を確かめれば採用できるかを示します。

2　検証ハーネスが制約する判断

前編「観測モデル」では、入力や負荷と実際の動作を結び付けました。検証ハーネスは、この記録を要求・設計から定めた適合条件に照らします。例えば、明細の取得結果と所定の負荷での性能を、同じ実行の記録で確かめます。

知識ハーネスはBoKで要求と設計の整合を確かめ、プログラミングハーネスは型やAPIの契約で実装を制約します。検証ハーネスは、実際に動いたプログラムの結果と処理の経過を確かめます。

表1は、変更の内容に応じた適合条件と、判断に使う実行記録を示します。

表1: 実装の適合条件と判定の根拠

制約の対象	適合条件	適合の根拠
業務上の結果	指定した入力で期待結果を満たす	実行結果と期待結果の照合
処理の分担と副作用	APIの契約に沿って処理し、設計で定めた副作用を保つ	呼び出し階層、処理の結果と状態の変化
所定の負荷での性能	レイテンシとスループットの目標を満たす	入力規模・リクエストミックス・負荷条件をそろえた実測
実行時に呼び出すAIモデルの構成	必要な生成品質と性能を満たし、費用の条件に収まる	同じ実行に対応する生成品質・処理時間・AIコスト

3　AI が生成した実装を実行時に検証する理由

実行時の検証が必要なのは、出力が正しくても、生成されたプログラムが設計で想定した経路や負荷を外れることがあるためです。実行基盤に任せるはずの再試行を補完コードにも書くと、失敗時に同じ処理が重複します。明細を一件ずつ読み込む実装では、結果が正しくても、明細数の増加に伴って I/O 回数が増え、性能目標を外れることがあります。

こうした問題の根拠になるのは、呼び出し元、呼び出し回数、処理区間ごとの時間です。実行条件と合わせて観測することで、負荷が集中したコードを絞り込み、修正後も守るべき結果や契約を明示できます。

計測やテストは従来から行われてきましたが、問題の修正を同じ条件で再検証し、採否を決めるところまでを日常の開発に組み込めていないケースも少なくなかったと考えます。SimpleModeling では、開発者が実行結果と CallTree の記録を見て、出力、呼び出し回数、処理時間が適合条件を満たすかを確かめます。その根拠をもとに、変更を採用するか判断します。

4　検証を支える標準機能

適合を判断するには、どの入力と条件で、どの実装がどう動いたかを追える根拠が必要です。
その根拠をそろえるために、CNCF 基盤と共通コンポーネントは次の四つを提供します。

- 実行データの自動採取 実行結果・処理時間・呼び出し階層を記録
- 観測モデルによる関連付け 実行条件と結果・計測値を対応付け
- Corpus による条件の固定 入力ケースと期待結果の版を固定
- Experiment による候補の比較 候補ごとの実行条件・結果・評価を比較

4.1　実行データの自動採取

CNCF は、Operation の実行結果と経過時間、内部 DSL を通る操作や I/O の呼び出し階層を自動採取します。CallTree で同じ操作が現れた経路をたどれるため、アプリごとの観測用コードを追加せずに、二重処理や I/O 反復の箇所を調べられます。

4.2　観測モデルによる関連付け

検証では、どの条件で得た結果や計測値かを特定できることが重要です。観測モデルは、Experiment の入力・実行条件・Arm・Run・結果・評価に、呼び出しの経路や回数、処理区間の時間を結び付けます。検証ハーネスは同じ Run の出力と時間を期待結果と性能目標に照らします。目標を外れた場合は、同じ記録から時間を要した処理と呼び出しの経路をたどり、修正の対象を絞り込みます。

4.3　Corpus による条件の固定

Corpus は、再実行する入力、期待結果、評価の根拠を版として固定したものです。問題を再現するケースと既存の代表ケースをそろえることで、修正前後を同じ起点から比較できます。性能を比べ

るときは、リクエストの種類・構成比・入力規模を定めたリクエストミックスも使います。

4.4 Experiment による候補の比較

Experiment は、候補と実行条件、結果と評価を結び付ける枠組みです。改修前後の実装や、プログラムが実行時に呼び出す AI モデルの構成を Arm とし、一回ごとの実行を Run として区別すると、同じケースで起きた差を候補ごとに比較できます。

5 変更の採否を実行のフィードバックに組み込む

検証ハーネスは、実測で見つけた問題を修正へ返し、適合を確認した変更を運用に戻すフィードバックループを構成します。

実運用の観測と Corpus による再実行を、次の四つの役割でつなぎます。

- 実運用の観測 実際の利用条件で異常と変更の効果を観測
- Corpus での障害再現 入力と負荷を固定して障害を再現
- 適合判定と変更の採否 実測を適合条件と照合し、採用・修正・追加確認を判断
- 修正の再検証 問題の解消と既存動作への影響を確認

図 1 は、実運用と Corpus の実行データを共通の原因調査と判定に使い、適合を確認した変更を運用に戻す関係を示します。

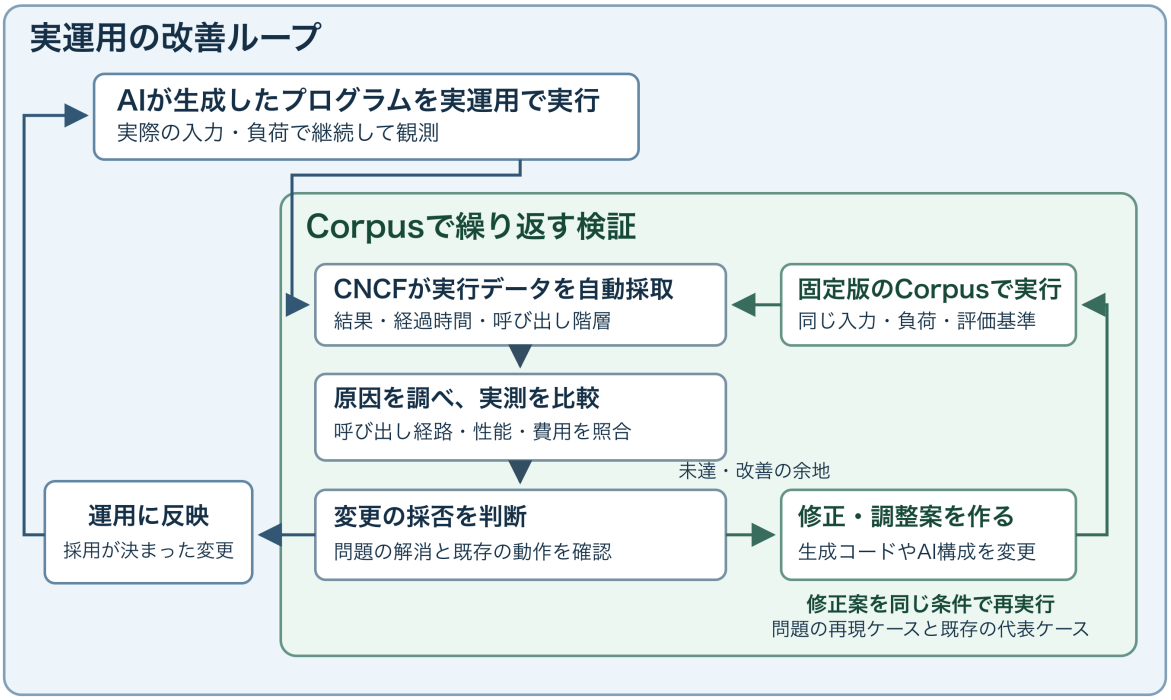

図 1 実運用の改善ループと、その中で回す Corpus による再検証

5.1 実運用の観測

実際の入力や負荷で初めて現れる問題は、運用中に蓄積した記録から調べます。CNCF が処理ごとに採取した実行結果、時間、呼び出し階層を見れば、起きた問題を再現用の入力や負荷条件に結び付けられます。採用した変更を反映した後も、実際の利用条件で効果を観測します。

5.2 Corpus での障害再現

障害を起こした条件とコードの変更が混ざると、実測の差を解釈しにくくなります。入力と負荷を固定して実行すれば、失敗を再現して原因を調べ、修正前後の差を同じ基準で判断できます。性能のチューニングでも、同じリクエストミックスでレイテンシとスループットを比較します。

5.3 適合判定と変更の採否

実行の結果が基準を満たす根拠を確認できたら、適合と判定して変更を採用します。基準未達の変更は、原因箇所の記録と維持すべき契約を添えて修正・再検証に戻します。判断に必要な根拠が不足している場合は、ケースや計測を補ってから採否を決めます。

5.4 修正の再検証

修正によって元の問題が解消しても、従来の入力で期待結果が変わることがあります。Corpus には障害を再現するケースと既存の代表ケースを含めます。前者で問題の解消を、後者で結果・副作用・性能が既存の条件に適合することを確かめ、両方の根拠を採否の判断に使います。

6 実行時に呼び出す AI モデルのチューニング

業務処理の中で AI を呼び出すプログラムでは、どの工程にどのモデルや推論設定を割り当てるかで、生成結果の品質、処理時間、AI コストが変わります。ここでいうチューニングは、この実行時の構成を調整して、必要な品質と性能を満たす構成で AI コストを抑えることです。

業務ロジックは共通にし、実行時の AI 構成ごとに Experiment の Arm を定義します。二つの候補なら A/B テスト、三つ以上ならマルチアーム検証として、同じ Corpus と評価基準で比較します。品質と性能の条件を満たした候補の中から、AI コストの実測に基づいて採用する構成を選びます。

7 注文書処理の採否を判断する例

注文書処理を例に、変更前後の動作や候補ごとの実測を、適合条件に照らして採否を決める方法を示します。

三つの変更について、検証する対象は次のとおりです。

- 二重再試行の修正 失敗時の重複呼び出しと正常時の結果を確認
- 明細 I/O の一括化 取得結果と副作用を保ったまま I/O を削減
- 実行時の AI 構成の比較 生成品質・性能・AI コストで採否を判断

7.1　二重再試行の修正

注文の登録先サービスが一時的にエラーを返すと、アプリケーションと実行基盤がそれぞれ再試行し、登録処理の呼び出しが重複するケースを考えます。呼び出し階層から二つの再試行経路を特定し、再試行を基盤に任せるようコードを修正します。障害を再現するケースで登録の重複が解消し、正常ケースでも注文の登録結果が期待どおりになることを確かめます。

7.2　明細 I/O の一括化

前編の例では、観測モデルに結び付けた CallTree から、1,000 個の明細の参照で 1,000 回の I/O が発行されたことを確かめました。ここでは、その処理を一括化した実装を採用できるかを検証します。改修前後を同じ版の Corpus と同じリクエストミックスで動かし、取得結果と副作用を保ったうえで、I/O 回数、レイテンシ、スループットが設計上の適合条件を満たすかを確かめます。

7.3　実行時の AI 構成の比較

注文書から項目を抽出し、解釈・補正して最終確認する処理では、各工程でプログラムが呼び出す AI モデルの組み合わせによって、生成品質と AI コストが変わります。この例では業務ロジックをそろえ、前編「観測モデル」の Arm A・B に、ローカル LLM を使う工程を増やした C を加えます。三つの構成を同じ Corpus の 100 ケース、最大同時実行数 4 で比較します。

この比較で適合を判断する条件は、次のとおりです。

- 生成品質 人の修正前に 100 件すべての必須項目が正解
- AI コスト 100 件合計の指数が 60 以下
- レイテンシ 受付から完了までの p95 が 8 秒以下
- スループット 最大同時実行数 4 で 40 件／分以上

表 2 は、各構成でプログラムが呼び出す AI モデルと推論設定を示します。

表 2: 注文書処理の各工程で呼び出す AI モデル（架空の構成）

Arm	抽出	解釈・補正	AI による最終確認
A	クラウド甲	クラウド甲・高推論	クラウド甲・高推論
B	ローカル LLM	クラウド乙・標準推論	クラウド甲・標準推論
C	ローカル LLM	ローカル LLM	クラウド乙・軽量モデル

表 3 は、各構成で人の修正前に必須項目がすべて正解だった注文書の件数、AI コスト、性能の実測を示します。AI コスト指数は A を 100 とし、人による確認・修正の負担は別に記録します。

表 3: 実行時の AI 構成ごとの生成品質・AI コスト・性能（仮想値）

Arm	**正解件数（人の修正前）**	**AI コスト指数**	**p95**	**スループット**
A	100/100	100	8 秒	30 件/分
B	100/100	45	6 秒	45 件/分
C	98/100	30	5 秒	60 件/分

B は生成品質・AI コスト・性能の条件を満たします。A は AI コストとスループット、C は生成品質が条件を満たさないため、この条件では B を採用します。

8 まとめ

観測モデルは、どの入力・負荷でプログラムがどう動いたかを、比較できる記録に整えます。検証ハーネスは、その記録を適合条件に照らして変更の採否を判断し、適合した変更を運用に戻す枠組みです。

実運用の観測と Corpus による再検証を、次の判断に使います。

- 生成コードの異常動作と性能低下を実測から検出
- 修正の効果と既存動作への影響を同じ Corpus で検証
- 実行時の AI 構成 生成品質と性能を満たす候補の AI コストを比較