



確率と戦うAIプロダクト

～「守り」と「攻め」を分ける2層テストアーキテクチャ～

JaSST'26 Niigata | 松山 晃大（株式会社LayerX バクラク事業部 QAマネージャー）

松山 晃大 @matsu

株式会社LayerX バクラク事業部 QAマネージャー

経歴

- 株式会社ヒューマンクレスト (2014/06 ~ 2015/12)
- 株式会社メルカリ (2016/01 ~ 2023/08)
- 株式会社LayerX (2023/11 ~)

趣味

- ポーカー、ゲーム、漫画、スノーボー



Agenda

目次

- 確率と戦うということ
- バクラク勤怠のAI機能
- 品質定義とリリース判断
- 2層テストアーキテクチャ
- 結合テスト層（守り）
- 単体テスト層（攻め）
- まとめ

「すべての経済活動を、デジタル化する。」をミッションに、複合的な事業を通して日本の社会課題を解決し、AIの力で人々の創造力がより発揮される未来をつくります。



バクラク事業

バックオフィス向け
AIエージェントサービスを提供



Fintech事業

資産運用サービス
「ALTERNA（オルタナ）」を提供



Ai Workforce事業

エンタープライズ向け
AIプラットフォームを提供



Security事業

AIエージェントによる
自律的なペネトレーションテストを提供

確率と戦うということ

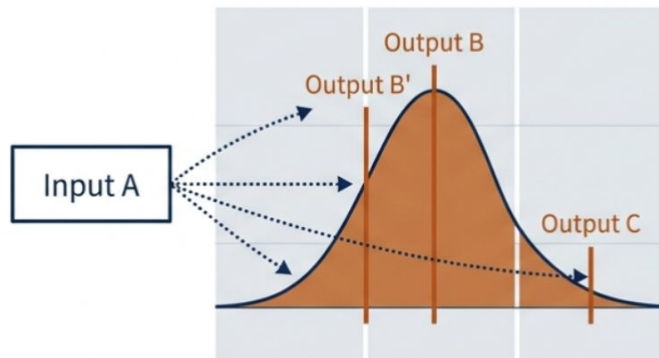
なぜ従来のテストのままでは守れないのか

生成AI時代の品質保証は「確率」との戦い

Traditional Software (決定的)



Generative AI (確率的)



品質保証の前提が、決定的なものから確率的なものに

確率的になったことによるテストの前提の変化

1回の「結果のブレ」と、時間の経過で起きる「性能の変化」。

同じ瞬間でも結果がブレる

同じ入力でも出力が揺らぐ。

1回の実行結果では、期待値と一致したかを合否として判断できない。

時間の経過で結果が変わる

LLMは同じモデルを使っている、毎日結果が変わる。

昨日はうまくいっていても今日うまくいくとは限らない。

一度テストをPASSしていれば品質が担保できるという前提が崩れた

pass/fail ではなく、スコアで判定する

複数回の結果をスコア化して判断する。

01

複数回実行する

1回の結果ではなく複数回の結果を見ることで、揺らぎを平均化する

02

成功率をスコアにする

項目ごとの成功率を、重要度に応じて重み付けして1つのスコアにする

03

基準を決める

リリース判断の受け入れ基準として、閾値をあらかじめ置いておく

リリース後も、精度を測り続ける

一度きりのテストではなく、継続的に精度を計り続ける仕組みが必要。

01

変更がなくても測る

こちらが何も変えていなくても精度は動くので、定期的に同じ基準で測り直す

02

差分で気づく

前回との差を見ることで、精度が落ちたことに気づける状態を保ち、揺らぎと退行を切り分ける

03

評価データを育てる

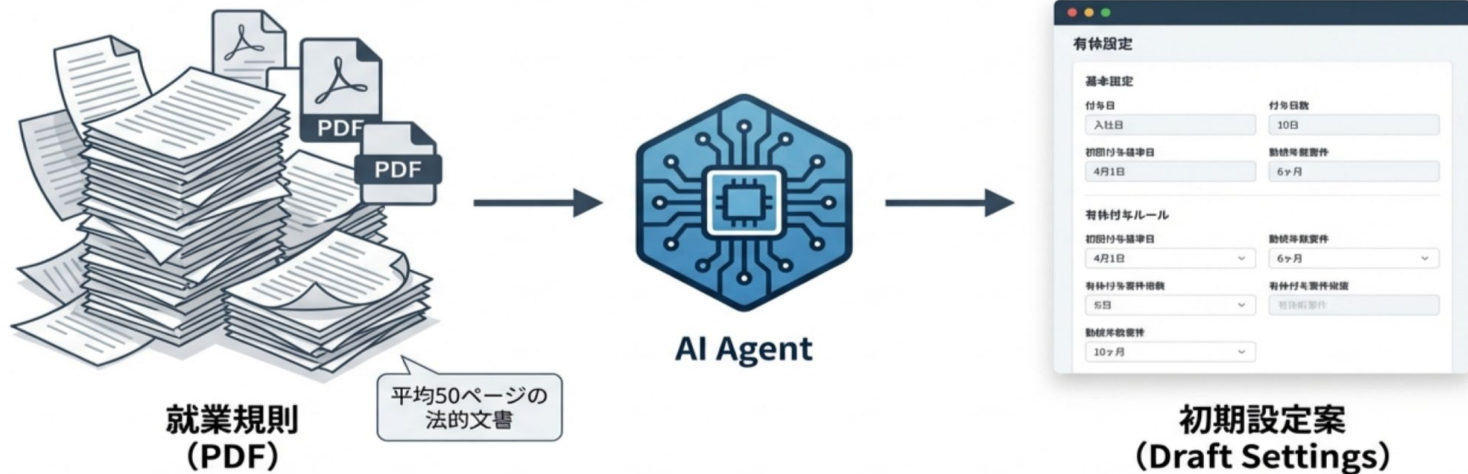
本番で崩れたケースを評価データに加え、次からは検知できるようにする

バクラク勤怠のAI機能

就業規則から有給付与ルールを作るAIエージェント

就業規則をもとに、AIが有給付与ルールを作成する

ユーザーは就業規則のPDFをアップロードするだけ。



有給の付与ルールは、就業規則を元に設定

付与する日数やタイミングは各社様々であり、フォーマットもバラバラ。

テキストだったり

第〇条（年次有給休暇）

1. 会社は、雇入れの日から起算して6か月間継続勤務し、かつ、当該期間の全労働日の8割以上出勤した従業員に対しては、10労働日の年次有給休暇を与えるものとする。
2. 会社は、前項の規定による年次有給休暇を付与した後、さらに1年間継続勤務し、かつ、前1年間の全労働日の8割以上出勤した従業員に対しては、雇入れの日から起算して1年6か月が経過した時点において11労働日の年次有給休暇を与えるものとし、以降も同様に、雇入れの日から起算して2年6か月間継続勤務し、かつ、直前の付与基準日から起算した過去1年間の全労働日の8割以上出勤したことが確認された従業員に対しては12労働日の年次有給休暇を与え、雇入れの日から起算して3年6か月間継続勤務し、かつ、同様に過去1年間の全労働日の8割以上出勤した従業員に対しては14労働日の年次有給休暇を与え、雇入れの日から起算して4年6か月間継続勤務し、かつ、同様に過去1年間の全労働日の8割以上出勤した従業員に対しては16労働日の年次有給休暇を与え、雇入れの日から起算して5年6か月間継続勤務し、かつ、同様に過去1年間の全労働日の8割以上出勤した従業員に対しては18労働日の年次有給休暇を与え、雇入れの日から起算して6年6か月以上継続勤務し、かつ、同様に過去1年間の全労働日の8割以上出勤した従業員に対しては、以降1年を経過するごとに20労働日の年次有給休暇を与えるものとする。ただし、前述の各期間において出勤率が8割に満たない場合には、当該年度の年次有給休暇は付与しないものとし、また、付与日数は20労働日を限度とする。
3. 第1項および前項の規定により付与された年次有給休暇の請求権は、これを付与された日から2年間行使しないときは、時効により消滅するものとする。

表だったり

第〇章 休暇等

採用の日より6ヶ月間継続勤務し、所定労働日の8割以上出勤した従業員に対して、6ヶ月を超えた日（これを応答日とする）に以下の表区分に応じた休暇を与える。

①社員・契約社員

継続年数	6ヶ月	1年 6ヶ月	2年 6ヶ月	3年 6ヶ月	4年 6ヶ月	5年 6ヶ月	6年 6ヶ月以上
付与日数	10日	11日	12日	14日	16日	18日	20日

②アルバイト・パートタイマー

			雇入れの日から起算した継続勤務期間の 区分に応ずる年次有給休暇の日数						
短時間労働者の週 所定労働時間	短時間労働者の週 所定労働日数	短時間労働者の1 年間の所定労働 日数(週以外の期間 によって労働日 数が定められている 場合)	6ヶ月	1年 6ヶ月	2年 6ヶ月	3年 6ヶ月	4年 6ヶ月	5年 6ヶ月	6年 6ヶ月 以上
			30時間以上						
30時間 未 満	5日以上	217日以上	10日	11日	12日	14日	16日	18日	20日
	4日	169日～216日	7日	8日	9日	10日	12日	13日	15日
	3日	121日～168日	5日	6日	6日	8日	9日	10日	11日
	2日	73日～120日	3日	4日	4日	5日	6日	6日	7日
	1日	48日～72日	1日	2日	2日	2日	3日	3日	3日

2. 付与の条件は応答日前日までの過去1年における所定労働日の出勤率が8割以上であることを要する。
3. 年次有給休暇は、特別の理由がない限り少なくとも5労働日前までに、所定の手続きにより届けなければならない。但し、業務の都合によりやむを得ない場合は、指定した日を変更することができる。

就業規則の中から、有給付与に関する条文を抜き出す

就業規則（平均50ページ）



ルールの設定項目（バクラク勤怠）

管理者設定 ▶ 有休付与ルール ▶ 新規作成 [ガイド](#)

有休付与ルール名 *

一斉付与 ☒ 一斉付与しない ☐ 一斉付与する 毎年 4/1 に付与

初年度付与タイミング ☒ 1回 入社6ヶ月後 に付与 ☐ 2回 (分割付与) 入社日 に付与 と 入社6ヶ月後 に付与

☐ 付与タイミングを月の1日に繰り上げる (入社日付とは除く)
2年目以降の付与も付与タイミングが繰り上がります

付与条件 ☒ 出勤率が80%以上の従業員のみ付与

毎年の付与日数

週の所定労働日数	初年	2年目	3年目	4年目	5年目	6年目	7年目以降
5日以上	10 日	11 日	12 日	14 日	16 日	18 日	20 日
4日	7 日	8 日	9 日	10 日	12 日	13 日	15 日
3日	5 日	6 日	6 日	8 日	9 日	10 日	11 日
2日	3 日	4 日	4 日	5 日	6 日	6 日	7 日
1日	1 日	2 日	2 日	2 日	3 日	3 日	3 日

難しい設定作業を、AIエージェントで解決したい

人が読み解いていた作業をAIに任せ、ユーザーはPDFを渡すだけ。

設定の手間をなくしたい

50ページの就業規則から該当条文を探して転記する作業を、AIが代わりに行う。

書き方の違いを吸収する

テキストでも表でも、会社ごとに異なる記述をAIが解釈し、設定項目に変換する。

AIに任せたいが、間違えられない領域でもある

AI有給付与ルールの実装

1つの大きなプロンプトで解かず、抽出 → 分割 → 判定 に役割を分けている。

1 条文を抽出する

`extract_sections`

PDF のテキストから有給休暇に関する条文を抜き出す。関連度の低いものは除外し、崩れた表は整形する

2 ルール単位を見抜く

`identify_rule_patterns`

雇用形態や入社月ごとに、設定上いくつかのルールに分かれるかを列挙する

3 詳細を判定する

6つのプロンプト

一斉付与・分割付与・出勤率・付与日数など6項目を、ルールごとに並列で判定する

プロンプトは全8種類。各ステップの出力は JSON スキーマで構造化している

品質定義とリリース判断

基準をどう決め、届かなかったときにどう判断したか

「品質の定義」の作成

QA4AI.Guidelinesを参考に品質を3つの軸に分けて定義した。

01

System Quality

システム全体の信頼性

従来のソフトウェアテスト（機能・性能・セキュリティ）に加え、AIの挙動を含めた受け入れ基準を定義する

02

Accuracy

精度を評価するための基準

評価の基準値と計算式を定義する。リリース時点で固定せず、運用のサイクルで育て続ける

03

Customer Expectation

ユーザーの期待値に対する仮説

「AIは100%完璧でない」という事実をユーザーと共有し、過度な期待を抱かせないように調整する

ref: [QA4AI.Guidelines](#)

実際に使用した品質定義

3つの軸をドキュメントに落とし、チームの共通認識にした。

System Quality

- ファイルアップロードと解析トリガー
 - ユーザーは就業規則（PDF）をアップロードできること。
 - アップロード後、解析処理が開始され、完了するまでユーザーに進捗（ローディング等）がフィードバックされること。
 - 解析不能なファイル（パスワード付きPDF、破損ファイル、文字が読めないPDF）の場合、適切なエラーメッセージが表示されること。
- ルールの抽出とオブジェクト生成
 - 就業規則内に「正社員」と「パート・アルバイト」など異なるルールが存在する場合、別々の有休付与ルールオブジェクトとして生成されること。
 - Accuracyで定義された精度が90%を超える
- 非機能要件・制約
 - ユーザーが「待てる」範囲（目安30秒～1分）で解析結果が返ってくること。タイムアウトしないこと。

Accuracy

- リリース前
 - 各ユースケースパターンを10回実行し、期待する結果になるかで精度の検証を行う
 -  就業規則/給与規定サンプル
 - CI/CDとして組み込みモデルの劣化など検知できる仕組みとする
 - 現状stgデプロイタイミングでのみ実行されているのでdaily実行を要検討
- リリース後
 - 1ヶ月程度運用して精度が閾値以下になったらアラートをあげる仕組みを作る
 - 運用してから分母/分子を決めてスコアを決定する
 - 初期はkarteでフィードバックをもらって、分母分子がある程度決まったらプロダクトに組み込む

Customer Expectation

- 仮説
 - AIによって出力されたルールが使われていないケースが期待値を満たさない
 - 逆にAIが出力したケースを全て使われていなくても一部でも使われていれば期待値を満たしていると言える

手動テストでの精度検証

自動化の前に、同じPDFを手動で何度も流してテストを行った。

10回

手動で実行する回数

同じパターンを10回流し、揺らぎを平均化して割合を見る

90%

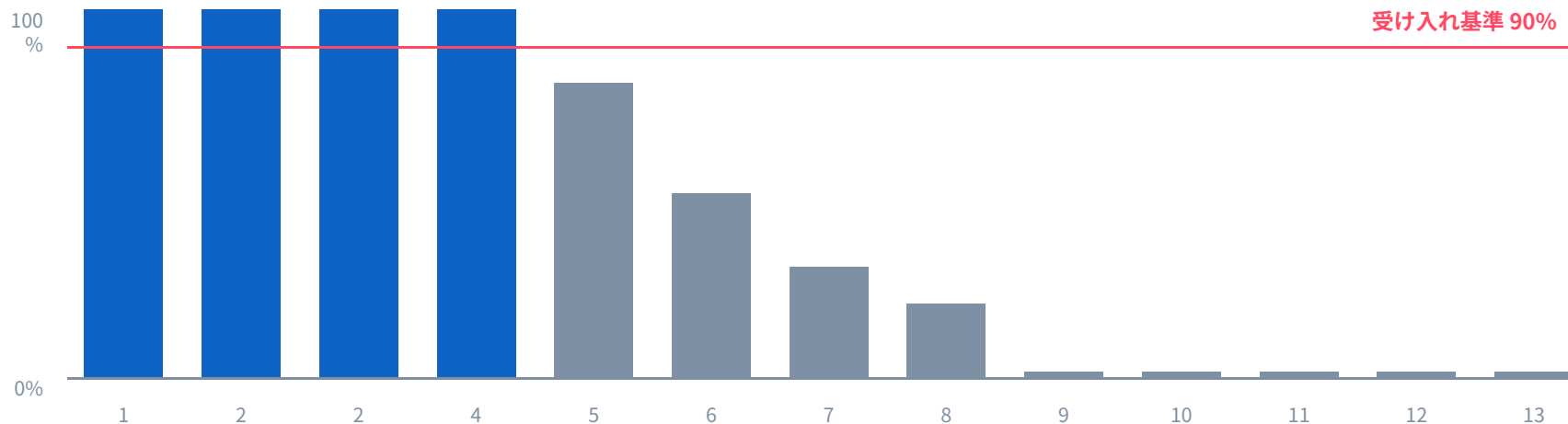
リリースの受け入れ基準

最終スコアがこれを下回ったらリリースしない

この時点では自動化されておらず、手動で測ってスコアを出していた

90% に届いたのは 13 パターン中 4 件だけ

就業規則のパターンごとに精度を検証。平均は 45%。



平均 45%。5 パターンは 1 件も正しく作れなかった

精度検証で分かったユーザーへの影響

ルールが2個以上の複雑な就業規則で精度が低いということがわかった。

ルールが1つ（全体の 2/3）

基本的にはカバーできている。ただし PDF に文字情報がない、表が特殊な記載の場合は外すこともある。

ルールが2個以上（全体の 1/3）

ものによってはカバーできるが、難しいケースは精度が落ちる。

リリースまで時間がないが精度が想定していたものに到底届かない

UX を自動作成から確認前提に作り替え

当初は確認画面なしで自動作成する設計。精度が届かず、確認して確定する形に変えた。

管理者設定
有休付与ルール
就業規則からAIで自動作成
確認

確認済 1 / 7
すべてのルールを確認してください

1 / 7
確認済み (作成する)
一斉付与(7月)_7月～12月入社者用
一斉付与 7/1
初年度付与 入社時
付与日数 法定通り

2 / 7
未確認
一斉付与(7月)_1月入社者用
一斉付与 7/1
初年度付与 入社時
付与日数 法定通りでない

3 / 7
未確認
一斉付与(7月)_2月入社者用
一斉付与 7/1
初年度付与 入社時
付与日数 法定通りでない

4 / 7
未確認
一斉付与(7月)_3月入社者用

このルールを作成対象にしますか？

☐ 作成対象にする
☐ 作成対象にしない
選択してください ※AIによる推奨設定です。誤りがないか十分にご確認ください。

編集

設定内容

・作成するかどうかを必ず回答
・「AIによる推奨設定」と明記

対象	一斉付与(7月)
一斉付与	する (毎年7/1に付与)
入社初年度付与	入社日に付与 (入社日基準を優先)
出勤率判定	行う
付与日数	法定通りでない

付与日

週の所定労働日数	初年	2年目	3年目	4年目	5年目	6年目	7年目以降
5日以上	8日	11日	12日	14日	16日	18日	20日
4日	6日	8日	9日	10日	12日	13日	15日
3日	4日	6日	6日	8日	9日	10日	11日
2日	3日	4日	4日	5日	6日	6日	7日
1日	1日	2日	2日	2日	3日	3日	3日

・項目の編集可

期待する精度は出せなかったが、リリースする判断を行った

精度

平均精度は 45% で、受け入れ基準の 90% には届かない。
プロンプトの改善だけではこの差を埋められなかった。

体験

自動作成から確認して確定する UX に作り替え、AI の出力
をサジェストに留めることで、設定の手間を減らせた。

体験そのものを変えれば、精度が期待に届かなくてもユーザー価値の提供はできる

2層テストアーキテクチャ

「守り」と「攻め」を目的で分ける

AI機能における品質保証の難しさと、我々のアプローチ

単一のテストスイートではなく、目的の異なる2つの層に分離した。

課題

- 予期せぬタイミングで精度低下が起きる
- 継続的な精度の改善も同時に必要になる

アプローチ

- 時間経過による性能の変化を検知し、精度を守るための結合テスト
- 今まで精度が高くなかったユースケースで精度を向上するための単体テスト

結合テスト層＝精度を落とさない「守り」 ／ 単体テスト層＝精度を高める「攻め」

なぜ1つのテストスイートで両方を兼ねないのか

目的が違うことでテストのスコープや頻度を変える必要がある。

■ テストスコープの違い

システム全体かプロンプト単位かの違い

■ 実行頻度の違い

定期実行か即時実行かのフィードバックサイクルの違い

■ 判定基準の違い

絶対評価か相対評価か

落ちたら困るところと、これから伸ばすところを、分けて判定することにした

テストの判定基準

結合テストと単体テストでそれぞれ基準を設定した。

90%

守り：絶対値で判定

同じ入力を複数回実行し、カテゴリ別の成功率を重み付けして算出。リリース時の受け入れ基準に揃えた。

-5%

攻め：main との差で判定

絶対値でなく main ブランチとの差分で判定。揺らぎの幅を許容しながら、精度の差分をチェックする。

守りは絶対値で下限を守り、攻めは相対値でデータを増やしやすくする

2層テストアーキテクチャの全体像

同じ機能に対して、目的の違う2種類のテストを別のサイクルで回す。

有給自動付与ルールのAI機能

壊れていないことを担保するガードレール

結合テスト層 (runn)

- 変更がなくても実施する
- システム全体の動作担保
- 精度確認済みの安定パターンを使う
- スコア < 90% でテストを失敗させる

精度を伸ばすための改善ループ

単体テスト層 (Go x Langfuse)

- 変更があった場合だけ実施する
- プロンプト単位での精度評価
- 本番で失敗したデータ・新規ケースを追加
- main との差 < -5% でテストを失敗させる

結合テスト層（守り）

runn で現状の品質を守る設計と、その運用

runn とは

YAML形式でシナリオを記述し、APIテストを自動化するためのオープンソースのCLIツール。



```
# 最終スコアを計算（重み付け: pattern 25%, basic 25%, one_to_four 10%, five_or_more 40%）
calculateFinalScore:
  bind:
    # 各カテゴリの成功率（デバッグ用）
    pattern_success_rate: (pattern_success_count * 100) / vars.total_count
    basic_success_rate: (basic_success_count * 100) / vars.total_count
    one_to_four_success_rate: (granting_days_one_to_four_success_count * 100) / vars.total_count
    five_or_more_success_rate: (granting_days_five_or_more_success_count * 100) / vars.total_count
    # 最終スコアを直接計算
    final_score: |
      (pattern_success_count * 25 +
       basic_success_count * 25 +
       granting_days_one_to_four_success_count * 10 +
       granting_days_five_or_more_success_count * 40)
      / vars.total_count
  assertFinalScore:
    desc: 'スコアレーティング'
    test: final_score >= vars.minimum_score
    dump: final_score
```

ref: [Github - k1LoW/runn](https://github.com/k1LoW/runn)

runn による結合テスト層

リグレッションテストの位置づけ。AI の精度だけでなく、システム全体を見る。

- **API にリクエストを送る** 実際のエンドポイントへリクエストを送信する
- **レスポンスを採点する** ステータスとデータ構造をもとにスコアリングする
- **絶対値で判定する** スコアが 90% を下回ったら失敗させる

AI の精度だけでなく、システム全体が正常に動くかを同時に確認できる

精度検証を行う流れ



$$\text{最終スコア} = (\text{パターン識別} \times 0.25) + (\text{基本情報} \times 0.25) + (\text{週1~4日} \times 0.10) + (\text{週5日以上} \times 0.40)$$

重み付けスコアリングの設計

業務インパクトの大きさに重みを決める。最終スコア 90% 以上で合格とする。

カテゴリ	重み	内容
パターン識別	25%	1つのPDFから何件のルールが抽出されるか
基本情報	25%	一斉付与・付与月オフセット・出勤率の使用など、基本の7項目
週1～4日の付与日数	10%	パート勤務者向け（32項目）。項目数が多いが影響範囲は小さい
週5日以上 の付与日数	40%	フルタイム勤務者向け（8項目）。法定付与日数の遵守が最も重要

テストの出力方法

カテゴリ別スコアと最終スコアを出力。

実際の出力結果

```
$ runn run integrations/attendance/ai_paid_holiday_rules/legal_standard_accuracy_evaluation.yml
```

```
[1/4] パターン識別 : 100%  
[2/4] 基本情報 : 100%  
[3/4] 付与日数 週1-4日 : 100%  
[4/4] 付与日数 週5日以上 : 100%  
[最終スコア] 100%
```

```
1 scenario, 0 skipped, 0 failures
```

カテゴリ別にどこかの精度が低いかを見せつつ、合否は最終スコアで判定する

CIの実行トリガーから結果通知まで



CIでの動作環境があることによって繰り返しテストを実行することが可能になっている

単体テスト層（攻め）

Go と Langfuse で精度を高める設計と、その運用

Langfuseとは

LLMアプリケーションのためのオープンソースな監視・評価プラットフォーム。



Langfuse v3.152.0 OSS ↑ layerone ▾ / attendance-test ▾

Go to... * K

Home

Dashboards

Observability

Tracing

Sessions

Users

Prompt Management

Prompts

Playground

Prompts ⓘ

Show filters Search... Names, Tags ⓘ ▾

Name	Versions	Type	Latest Version Created At ▾	Number of Observations
identify_rule_patterns	5	text	2026-01-30 11:32:40	126
split_grant	4	text	2026-01-28 22:26:33	199
simultaneous_grant	4	text	2026-01-28 22:26:32	199
excluding_joining_date	4	text	2026-01-28 22:26:32	199
attendance_rate	5	text	2026-01-28 22:26:32	245
annual_grant_days	10	text	2026-01-28 22:23:00	466
grant_priority	3	text	2026-01-08 16:44:37	199
extract_sections	3	text	2026-01-08 16:44:05	126

ref: [Langfuse Overview](#)

Go x Langfuse による単体テスト層

新機能テストの位置づけ。各プロンプトの精度を改善するためのテスト。

- **変更があった場合のみ実行する** プロンプトの変更を検知し、自動で実行される
- **プロンプト単位で実行する** プロンプトごとに実行し、出力を検証する
- **相対値で判定する** main との差が -5% を下回ったら失敗させる

修正が入った時に修正があった箇所をテストできる仕組み

データは Langfuse、判定は Go と役割を分けている

Langfuse では監視・データを持たせ、採点と判定は Go 側に置いた。

Langfuse

評価の材料を持つ

本番実行に使うプロンプトのバージョン、評価用データセット、実行トレースを持つ。

Go

実行して採点・判定する

データセットを取得してプロンプトごとに採点。PR 版と main 版を比べて判定する。

プロンプトの修正は、アプリのリリースを待たずに反映できる

変更したプロンプトの自動評価

変更のあったプロンプトだけを、PR 版と main 版の両方で実行して比べる。

```
if (PR Score - Main Score) < -0.05 then FAILED
```

変更したプロンプトだけ回る

PR で変更されたプロンプトを git diff で拾い、そのプロンプトだけを評価する。

main と同じ条件で比べる

main 版のプロンプトも同じデータセットで実行し、差分で判定。結果は PR にコメントされる。

GitHub ActionsによるPR単位のフィードバック

PR のスコアが main より 高いと成功し、5% 以上下がるとチェックが失敗する。



github-actions bot commented 2 weeks ago



Prompt Evaluation Results

1 個のプロンプトを評価しました:

 Prompt: **identify_rule_patterns**

✅ 評価完了 (PR 版 vs Main 版 比較)

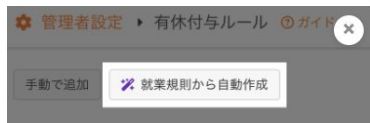
メトリクス	PR 版	Main 版	差分	判定
平均スコア	100.000	98.000	+2.000	🌟 向上
成功率	100% (9/9)	66.66666666666666% (6/9)	+33.3%	(参考)

▶ PR 版詳細結果

▶ Main 版詳細結果

実例① ユーザーにアンケートを出す

有給付与ルール of 自動作成を行ったユーザーに、利用の直後に KARTE でアンケートを表示する。



AI自動作成機能はいかがでしたか？

Q1. 設定は完了しましたか？ *

はい

いいえ

Q2. 満足度はいかがですか？ *



満足



イマイチ

Q3. ご意見・改善点

気になった点があればご記入ください

必須項目を入力してください



KARTE2Slack通知 アプリ 16:33

有休付与ルールのアンケート回答がありました 📝

満足度

bad

コメント

【設定:未完了】

全てのパターンが作成されなかった

KARTE User Story

管理画面でユーザーストーリーを見る



1



実例② トレースで原因を分析する

アンケートの回答をもとに、Langfuse のトレースでどのプロンプトの出力がズレたのかを特定する。

The screenshot displays the Langfuse interface for analyzing a trace. On the left, a tree view shows the sequence of steps: ExtractRuleFromPDF, ExtractRuleFromPDF, ExtractStructuredSections, **extract_sections** (selected), IdentifyRulePatterns, identify_rule_patterns, 社員用, excluding_joining_date, grant_priority, annual_grant_days, simultaneous_grant, attendance_rate, and split_grant. The main panel shows details for the 'extract_sections' step, which occurred on 2026-02-06 at 13:14:06.879. It includes metrics for latency (7.77s), environment (default), cost (\$0.081906), and completion (35,077 prompt → 1,469 completion). The prompt is 'extract_sections - v2'. The 'Preview' tab shows the 'Input' and 'Output' sections. The 'Output' section shows a list of items, with one item highlighted in green. The 'Corrected Output (Beta)' section shows a placeholder for a corrected output. The 'Metadata' section shows a list of items.

実例③ データセットに追加する

トレースを分析し、期待値を修正して追加すると、そのままテストケースになる。

Dataset identify_rule_patterns						+ New item
Runs Items						
Search... IDs Filters						
Item id	Source	Status	Created At	Input	Expected Output	
pozmggyiyc1aeqeyu...		Active	2026-01-19 15:52:54	<pre>{ 1 Items sections: [3 Items 0: { 3 Items content: "従業員の年次有給休暇は、入社月に じた第2項の表の区分により毎年1月1日又は7月1日</pre>	<pre>{ 2 Items reasoning: "就業規則第39条第2項において、入社月 (1月~12月) ごとに個別の初回付与日数が定義されて いるため、これらをシステム上の仕様に合わせて12個の ルールに分割する。また、基準日が1月1日または7月1日</pre>	
qa9lkkh5rgkeir5rcs7z...		Active	2026-01-19 16:03:25	<pre>{ 1 Items sections: [3 Items 0: { 3 Items content: "会社は、毎年4月1日から翌年3月31日 までは休暇年度とし、毎年4月1日に前年度の所定労</pre>	<pre>{ 2 Items reasoning: "就業規則の第61条において、年次有給休 暇の付与日数は入社月によって明確に区分されています (4~9月入社: 10日、10月入社: 8日など)。システム の設定上、初年度の付与日数が異なる場合は別のルール</pre>	
tqhwtlwnxu77quvmw...		Active	2026-01-19 15:52:40	<pre>{ 1 Items sections: [3 Items 0: { 3 Items content: "1 従業員のうち契約社員以外に対する 年次有給休暇は、毎年8月1日を基準日とし、次の表</pre>	<pre>{ 2 Items reasoning: "就業規則の条文では、雇用形態ごと（正 社員、短時間正社員・嘱託・無期転換、契約社員）に付 与テーブル・基準日・付与タイミングが明確に分かれて いるため、雇用形態ごとにルールを分割する必要がある</pre>	
u2rr6kh1gk3ex4z8a1...		Active	2026-01-29 16:37:12	<pre>{ 1 Items sections: [3 Items 0: { 3 Items content: "1. 6ヶ月間継続勤務し、所定労働日数 の8割以上出勤した者には、10日の年次有給休暇を</pre>	<pre>{ 2 Items reasoning: "就業規則の条文では、雇用形態による区 分や入社月による区分は明記されておらず、全社員に対 して同一の付与基準が適用されている。所定労働日 数・時間による比例付与は1つのルール内のテーブルで</pre>	

ユーザーフィードバックを起点にテストケースを増やす

1 KARTEでアンケート

有給付与ルール of 自動作成を行ったユーザーに、利用の直後にアンケートを表示する



2 トレースで原因分析

Langfuse で、どのプロンプトのどの出力でズレたのかを特定する



3 データセットに追加

実データから期待値を修正して追加。そのままテストケースになる

このループを回すことでテストを増やし、精度を改善させていく

まとめ

実際に運用して見えてきたこと

よかったところ

結合テスト層では、精度以前の「そもそも動作しない」レベルの不具合を発見できた。単体テスト層では、相対的な評価をすることによってデータの追加とプロンプト改善の分離ができ、改善のサイクルが回しやすくなった。

うまくいかなかったところ

想定よりも揺らぎの幅が広く、期待値の見直しやデータの手入れが絶えず発生し、メンテナンスコストは高い。

品質保証の範囲は広がったが、運用コストとのバランスはこれからの課題

2層テストアーキテクチャのまとめ

目的が違えば、実行タイミングもスコープも判定基準も変わる。

項目	結合テスト (runn)	単体テスト (Go x Langfuse)
実行タイミング	定期実行	プロンプトを変更した PR で実行
テストスコープ	システム全体の結合(API 経由の一連の動作)	プロンプト単位(1つの入出力)
テストデータ	精度が高いと確認済みのパターン	本番で失敗したデータ・新規ケース
判定の仕方	絶対値：最終スコアが基準を下回ると失敗	相対値：main とのスコア差が下がると失敗

ご清聴ありがとうございました