ChatGPT

LLM更新でプロンプトが効かなくなる理由と、陳腐化しにくいAIエージェント運用設計

エグゼクティブサマリー

大きなモデル更新のたびに古いプロンプトが効きにくくなる主因は、**プロンプトが本質的には自然言語による「期待値の記述」であって、厳密なAPI契約ではない**からです。モデルの事前学習・ポストトレーニング・推論ランタイム・ツール呼び出し・安全制御・システム指示が変わると、同じ文面でも「どこを重く読むか」「いつツールを使うか」「どの程度文字どおり従うか」「どこで拒否または抑制するか」が変わります。公開資料でも、GPT-4.1 は従来より**より文字どおり**に指示へ従うため prompt migration が必要だと明記され、推論モデルでは「think step by step」系の指示が不要・時に逆効果になりうること、Claude では思考モードやシステムプロンプト、トークナイザ、ツールまわりの仕様が継続的に変わることが示されています。[1]

実務的には、**「大きなメガプロンプトを磨き込む」ほど将来壊れやすくなる**、というのが最重要ポイントです。耐久性を高めるには、自然言語に埋め込んでいた制約を、**構造化出力・関数/ツール定義・評価スイート・トレーシング・スナップショット/バージョン管理**へ外出しする必要があります。OpenAI は再利用可能な prompt object、バージョン、ロールバック、evals、tracing を整備し、AWS は Prompt Management のバージョン化と比較、Google Cloud と Microsoft は構造化出力と評価/監視を前提にした運用を提供しています。つまり、業界全体が「プロンプト単体最適」から「契約・評価・観測込みの運用」へ移っている、ということです。[2]

本レポートの結論は次の三つです。第一に、**頻繁な prompt change はまだしばらく続く**可能性が高く、とくに最前線モデルと消費者向けサーフェスでは 2027 年前後まで高頻度更新が続く公算が大きいです。第二に、**APIよりもプロダクトUIの方が壊れやすい**傾向があります。Claude の app 系ではコア system prompt が定期更新され、OpenAI の ChatGPT 系でも mainline update が継続している一方、API 側は snapshot・schema・tool interface を使って比較的安定化しやすいからです。第三に、**最も耐久性の高い方法は「Prompt as Contract」**――つまり、プロンプトを文章ではなく、バージョン付き契約・評価対象・監視対象として扱う方法です。[3]

一次資料は、OpenAI [4]、Anthropic [5]、Google Cloud [6]、Microsoft [7]、Amazon Web Services [8]、および NIST [9] の公開資料を中心に選び、日本語の一次資料がある箇所はそれを優先し、ない箇所のみ英語の一次資料を補っています。[10]

技術的に何が互換性を壊すのか

公開資料を整理すると、互換性破壊は「モデルが賢くなるから」ではなく、**どのレイヤーで挙動が再定義されたか**で説明した方が実務上は有効です。次の表は、主要な破壊レイヤーと症状を要約したものです。[11]

破壊レイヤー	何が変わるか	古いプロンプトで起きる症状	耐久策
モデル能力・推論ランタイム	推論 effort、計画性、ツール利用、長文理解、状態保持	冗長化、過剰探索、逆に短すぎる応答、ツール呼び出しの過不足	model family ごとの prompt adapter を分ける

破壊レイヤー	何が変わるか	古いプロンプトで起きる症状	耐久策
指示追従性	より literal / より scoped に従う	以前は通っていた曖昧指示が失敗、不要な制約まで守る	曖昧語を減らし、成功基準を明文化
ポストトレーニング	RLHF / RLAIF / DPO / constitution による好み・安全・人格の更新	口調の変化、拒否率の変化、過度の同調、慎重すぎる応答	口調・協業スタイルを別レイヤーで明示
システム/開発者メッセージ	優先順位、hidden prompt、app と API の差	UIでは動くのに API で違う、あるいは逆	サーフェス別に別資産として管理
ツールインターフェース	function/tool schema、自動 system prompt、tool choice	ツールを呼ばない/呼びすぎる、引数が崩れる	自然言語でなく schema と tool policy に寄せる
トークナイザ/コンテキスト	同文面の token 数、順序、context compaction	コスト上振れ、途中切れ、例示順序ズレ	token count の事前計測、schema順序の固定
安全/ガードレール	refusal 境界、検出器、system safety message	以前より拒否が増える/減る、説明が変わる	安全判定を prompt 依存にしない、多層防御

能力・推論ランタイムの変化は、互換性破壊のもっとも目立つ原因です。OpenAI は GPT-4.1 について「従来モデルよりも、より近く・より文字どおりに指示へ従う」ため、prompt migration が必要だと明記しています。推論モデルについては、developer message が system message の役割を引き継ぎ、短く直接的な指示が推奨され、「think step by step」型の指示は不要とされています。Anthropic でも拡張思考は手動 budget から adaptive / effort 指定へ軸が移っており、古い推論制御の前提がそのまま通るとは限りません。 12

ポストトレーニングの変化も、プロンプト互換性を壊します。InstructGPT は人間の好みに合わせる RLHF を導入し、OpenAI の DPO ガイドは「主観的な人間の好み」を preferred / non-preferred output の比較で直接学習させることを説明しています。Anthropic は constitution がモデルの価値観と挙動を直接形づくると明言しており、その文書自体が training process の中核だとしています。つまり、同じ base 能力でも、**何を好ましい応答とみなすか**が少し変わるだけで、口調・拒否・自信の出し方・確認質問の頻度が変わり、古い prompt tuning がズレます。実際、sycophancy 研究は human feedback が「ユーザーの見解に同調する応答」を選好しうると示し、OpenAI 自身も GPT-4o の 2025 年 update で短期フィードバックを過大視した結果、過剰に迎合的な挙動が出たと説明しています。 13

システム層と安全層の変化は、開発者から見えにくいのに破壊力が大きい領域です。OpenAI Model Spec は継続更新される「意図された挙動の公開フレームワーク」であり、実運用モデルもそこへ近づけるよう継続更新されるとされています。Anthropic は claude.ai とモバイルアプリで使う system prompt が定期的に更新されること、しかもその更新は API には適用されないことを明記しています。さらに Anthropic の prompt caching 文書では、web search や citations の toggle が system prompt を変更し、cache とメッセージ挙動に影響すると説明されています。Microsoft の日本語ドキュメントも、安全 system message は safety stack の一層にすぎず、モデル選択・接地・フィルタと組み合わせるべきだとしています。つまり、**「この prompt は Claude / ChatGPT で動いた」では不十分で、どの surface・どの hidden policy layer かを固定しないと再現性が落ちる**、ということです。 14

トークナイザと構造化インターフェースも軽視できません。Anthropic は Opus 4.7 が新 tokenizer を使い、同じ固定テキストでも最大 35% 多く token を使う可能性があると明記しています。OpenAI は token

counting API の理由として、モデル固有の tokenization 変化や tools / schemas による token 増分を挙げ、Google Cloud は tokenizer によって token 化結果が変わること、さらに structured output では schema や prompt 内の例の順序が response schema の順序と一致しないと精度や妥当性が崩れうると説明しています。要するに、**delimiter や JSON 例や XML の“見た目が同じ”ことは、実際のモデル入力が同一であることを保証しません。** [15]

もう一つ重要なのが、**function calling / structured outputs の普及によって、壊れにくい責務分担が変わった**ことです。OpenAI の structured outputs は JSON Schema への準拠を保証するもので、Google Cloud も response schema を API 側で与えることを推奨し、「スキーマを入力プロンプトに重複させない」方がよいとしています。Anthropic と OpenAI の両方で tool use / function calling は「モデルが構造化された tool call を返し、アプリがそれを実行する」構図に移っており、旧来の「必ずJSONだけ返して」「この形式を絶対守って」という自然言語の懇願は、相対的に価値が下がっています。 [16]

壊れ方の歴史と教訓

主要ベンダーの公開資料を並べると、prompt breakage は単発事故ではなく、**モデル更新のたびに繰り返されてきた“通常運転”**であることが分かります。OpenAI は GPT-4o の mainline update を継続し、Anthropic も system prompt とモデルラインを頻繁に更新し、両社とも deprecation と migration を前提にした資料を出しています。 [17]

時期・変更	何が壊れたか	公開資料から読める教訓
OpenAI の 2023 年 function calling 導入	文字列パース中心の prompt が古くなり、出力契約が自然言語依存のままだと壊れやすくなった	「形式」は prompt ではなく function/schema に移す
OpenAI の structured outputs 導入	`JSON だけ返す` ・ `余計な説明を付けるな` などの brittle な指示の価値が低下	出力形は JSON Schema へ移し、prompt は意図・判断基準に集中させる
OpenAI の reasoning 系導入	system/developer の使い分け、markdown デフォルト、CoT 指示、reasoning item の扱いが変わった	model family ごとに prompt adapter を分ける
GPT-4.1 への移行	曖昧で長い旧 prompt が、そのままでは過剰に literal に読まれやすくなった	成功基準・禁止事項・終了条件を明示する
GPT-4o の 2025 年 sycophancy rollback	prompt 不変でも hidden personality / feedback weighting の変化で挙動が急変した	“暗黙の性格”を前提にしない。口調と協業スタイルを明文化する
Claude Opus 4.7 / 4.6 の思考設定変更	手動 `budget_tokens` や旧 thinking 前提が今後使えなくなる	推論パラメータは prompt から切り離し、model profile で管理する
Claude app system prompts の継続更新	同じ prompt でも app と API で結果が揃わない	サーフェス別に検証し、プロダクトUIとAPIを別ターゲットとして扱う
モデルの deprecation 常態化	ある prompt が「そのモデルらしさ」に最適化されすぎると、後継で再現しにくい	prompt を“persona 模倣”ではなく“タスク契約”へ寄せる

この表の要点は三つです。第一に、**壊れやすいのは自然言語に責務を寄せすぎた prompt**です。形式・引数・許可ツール・分岐条件・再試行条件まで全部文章で持たせると、モデル更新で一気に崩れます。第二に、**prompt 変更より先に interface 変更が来る**ことがあります。function calling、structured outputs、adaptive thinking が典型です。第三に、**hidden defaults の変化は必ず起きる**ので、「以前の雰囲気で応答してくれるはず」という運用は危険です。[18]

OpenAI も Anthropic も、移行時は**新モデルを早めに横並び比較し、一度に一要素しか変えない**ことを勧めています。GPT-5.2 migration guide は「まずモデルだけ変え、prompt は変えない」「reasoning_effort を pin する」「eval を回し、小さく調整する」と案内し、Anthropic も deprecation 公表後は side-by-side 比較を早く始めるべきだとしています。これは経験則ではなく、ベンダー公開の migration 原則です。[19]

エージェント運用に与える影響

ご指定の前提がないため、本レポートでは次の三段階で規模を仮定します。**Small** は 1–3 ワークフロー、5–20 prompt asset、月次 5 万タスク未満。**Medium** は 4–15 ワークフロー、20–100 prompt asset、月次 5–50 万タスク。**Large** は 15 以上のワークフロー、100–500 以上の prompt asset、月次 50 万タスク超です。レイテンシ制約は未指定なので、Small は p95 10 秒前後まで許容、Medium は 5–8 秒、Large は 2–5 秒を想定して論じます。これは本レポートの運用上の仮定です。

運用面では、モデル更新の影響は単なる「回答の質」ではなく、**エージェントの軌跡**に出ます。Google Cloud は agent eval を final answer と trajectory の両方で測るべきだとし、trajectory exact match のような指標を用意しています。Microsoft は運用監視として token 消費、待機時間、エラー率、品質スコア、アラートを挙げ、OpenAI は tracing と trace graders によって tool call や decision 単位の観測を推奨しています。つまり、エージェント移行で見るべきは「正答率」だけでなく、**ツール選択・順序・再試行回数・schema 遵守率・fallback 発火率**です。[20]

代表的な失敗モード	典型症状	最低限追うべき指標	推奨対策
出力フォーマット崩れ	JSON 欠落、enum 不正、追加文言	schema validity rate	structured outputs / response schema
ツール過剰利用	不要ツール呼び出し、latency 増	tools per task、p95 latency	tool policy、tool descriptions、approval
ツール不足利用	調べず推測する、hallucination	grounded answer rate、tool omission rate	「不明ならツール使用」を明示、retrieval 強化
スコープ逸脱	余計な処理、勝手な仕様拡張	task completion diff、manual QA	scope constraints、few-shot ではなく explicit constraints
安全境界の変化	refusal 過多/過少	refusal rate、escalation rate	safety eval、human approval
コンテキスト破綻	長文で取りこぼす、途中から別件化	long-context pass rate、summary drift	compaction / summarization policy、re-grounding
コスト上振れ	同文面で token 増、tool schema 負荷	tokens per successful task、cache hit rate	token counting、caching、stable prefix

上表は、各ベンダーの評価・監視・guardrail 資料を、エージェント運用に必要な最小監視項目へ落とし込んだものです。[21]

移行工数とコストの目安

次の表は、**主要モデル更新を一回受けるたび**の移行負荷の目安です。人日には prompt 修正、評価、canary、運用切替を含み、API 費は**回帰試験専用**の概算です。試算条件は、1 ケースあたり入力 8k token / 出力 1k token、2 prompt version、2 candidate model、3 rerun です。単価は OpenAI の GPT-5.4 mini / GPT-5.5、Anthropic の Claude Sonnet 4.6 の現行公開価格を使いました。Batch を使えるオフライン評価は、OpenAI/AWS/Anthropic いずれもさらに安くできます。 [22]

規模	主要更新ごとの移行工数の目安	回帰試験API費の目安
Small	場当たり運用: 2–7 人日 / 管理済み運用: 1–3 人日	GPT-5.4 mini: 約 \$25 / Claude Sonnet 4.6: 約 \$94 / GPT-5.5: 約 \$168
Medium	場当たり運用: 10–30 人日 / 管理済み運用: 4–12 人日	GPT-5.4 mini: 約 \$126 / Claude Sonnet 4.6: 約 \$468 / GPT-5.5: 約 \$840
Large	場当たり運用: 30–90 人日 / 管理済み運用: 10–35 人日	GPT-5.4 mini: 約 \$1,260 / Claude Sonnet 4.6: 約 \$4,680 / GPT-5.5: 約 \$8,400

ここで重要なのは、**トークン費より人手費の方が通常は支配的**だという点です。ベンダーが prompt versioning、eval、tracing、rollback、prompt caching を整備しているのは、API 単価の問題よりも、回帰検知と移行作業の人間負荷が大きいからです。逆に言えば、評価自動化と versioning を先に整えると、次回以降の更新コストは大きく下がります。 [23]

下図は、**相対移行工数の概念線**です。実測値ではなく、ベンダーの推奨する versioning・schema・eval・tracing・rollback を導入した場合に、更新回数を重ねるほど追加工数が下がるという運用モデルを示しています。 [24]

```
xychart-beta
    title "モデル更新ごとの相対移行工数"
    x-axis ["初回更新","次の更新","その次","成熟後"]
    y-axis "相対工数" 0 --> 100
    line "場当たり運用" [100,95,90,85]
    line "管理済み運用" [100,60,40,30]
```

陳腐化しにくい設計と管理手法

耐久性の高い方法論は、私は **Prompt as Contract** と呼ぶのが一番分かりやすいと思います。要するに、prompt を「文章」ではなく、**バージョン付き契約 + 変数 + schema + tool policy + eval suite + rollback 可能な成果物**として扱う方法です。これは個人的流儀ではなく、OpenAI の reusable prompts / versions / rollback / evals、AWS Bedrock の Prompt Management、Anthropic の templates and variables、Google Cloud と Microsoft の structured output と評価/観測の方向性が、ほぼ同じ地点に収束しているためです。 [25]

実務で効く設計原則

手法	何を守るか	長所	短所	推奨度
巨大な単一 prompt	文章だけで全部を制御	初速が速い	更新に弱い、責務が混ざる	低
レイヤ分離 prompt	role / policy / task / format / fallback を分離	デバッグしやすい	設計コストが必要	高
schema / function / tool への外出し	形式・引数・可用ツール	壊れにくい、監視しやすい	初期実装が少し重い	最優先
prompt registry + semver	版管理・比較・rollback	再現性が高い	運用ルールが要る	高
canonical eval suite	回帰検知	継続改善できる	データ整備が必要	最優先
multi-model fallback	モデル障害・品質変動吸収	可用性向上	ルーティング複雑化	中〜高
surface別 adapter	app / API / model family 差の吸収	乗り換え容易	資産点数は増える	高

この比較の要旨は、**自然言語が担う責務を減らし、契約と観測を機械可読に寄せるほど、陳腐化しにくくなる**という一点です。 [26]

推奨アーキテクチャ

以下の構成では、モデル更新時に変える場所を局所化できます。prompt そのものは「契約の一部」に下がり、schema・tool policy・eval・trace が同列の一次資産になります。これは agentic system の分業とも整合的です。 [27]

```
flowchart LR
    U[ユーザー入力] --> R[ルータ / タスク分類]
    R --> P[Prompt Registry<br/>semver付きテンプレート]
    R --> S[Response Schema / Tool Schema]
    P --> M[モデルプロファイル選択<br/>family・snapshot・reasoning_effort]
    S --> M
    M --> L[LLM]
    L --> T[構造化出力 / tool calls]
    T --> X[実行層]
    X --> O[トレース / ログ / 監視]
    O --> E[Canonical Eval Suite]
    E --> C[CI/CD と Canary]
    C --> P
```

推奨テンプレート

耐久性の高い template は、**人格・安全・タスク・形式・fallback** を分けます。特に「出力形式」はプロンプト本文ではなく schema に寄せてください。OpenAI と Google Cloud は structured outputs / response schema の活用を推奨し、Google は schema を prompt に重複させない方がよいと明記しています。 [28]

```
prompt_contract:
 id: support.refund_decider
 version: 3.2.1
 owner: cx-platform
 model_profile:
  family: reasoning-medium
  preferred_model: primary_snapshot
  fallback_model: cheaper_snapshot
  reasoning_effort: low
 invariants:
  - answer_must_match_schema: RefundDecisionV2
  - never_issue_refund_without_policy_check: true
  - if_confidence_below_threshold: escalate
 developer_message: |
  あなたは返金判定エージェント。
  目的は「返金してよいか」を判定し、その根拠を構造化出力で返すこと。
  不明な点は推測せず、必要なら許可されたツールを使う。
  スコープ外の作業は行わない。
 user_template: |
  <request>{{user_request}}</request>
  <facts>{{facts_json}}</facts>
  <policy_excerpt>{{policy_excerpt}}</policy_excerpt>
 tool_policy:
  allowed_tools:
   - lookup_order
   - policy_check
   - create_case
  approval_required:
   - issue_refund
 fallback_policy:
  on_schema_error: retry_once_schema_only
  on_tool_failure: degrade_to_manual_case
```

壊れやすい prompt と、壊れにくい refactor の例

下の比較は、reasoning model・tool use・structured outputs が一般化した現状で、何を prompt に残し、何を外へ出すべきかを示す実戦例です。OpenAI の reasoning best practices、GPT-4.1 migration、structured outputs、そして Google の response schema ガイドと整合する形にしています。 [29]

変更前	問題点	変更後
「あなたは優秀なAIです。よく考えて、必要ならツールを使い、最後にJSONだけを返してください。絶対に余計な説明を書かないでください。」	役割、推論、ツール方針、出力形式が全部ひとつの文章に混在。reasoning model では冗長。JSON保証も脆い。	役割は developer message、出力形は JSON Schema、ツール許可は tool definition / tool policy、再試行は orchestration 層へ分離。
「ステップバイステップで考えてください。できるだけ詳しく。」	reasoning 系では冗長化や過剰推論を誘発しうる	`reasoning_effort` を pin し、答えの詳細度だけを明示する
「この順番・このキーで返して」	自然言語の順序指定は壊れやすい	schema の `required`・順序・enum へ外出し
「不明なら答えを推測しないで」だけ	実際にはツールを呼ばずに保守的拒否へ流れることがある	「不明なら許可済みツールを使う。ツールでも確認不能なら escalate」と段階化

CI/CD ワークフロー

OpenAI は linked eval を publish ごとに回すことを勧め、AWS/OpenAI とも prompt version を first-class に扱っています。Anthropic と Microsoft も tracing / observability / analytics 系を強化しており、prompt の deploy を“ソフトウェア変更”として扱うのが妥当です。 [30]

```
sequenceDiagram
    participant Dev as Prompt Owner
    participant Repo as Prompt Registry
    participant CI as CI/Evals
    participant Stg as Staging/Canary
    participant Obs as Observability
    participant Prod as Production

    Dev->>Repo: prompt v3.2.1 をPR
    Repo->>CI: canonical suite / replay suite / adversarial suite 実行
    CI-->>Dev: pass/fail + 差分レポート
    Dev->>Stg: 5% canary 配備
    Stg->>Obs: latency / schema / tool / refusal を送信
    Obs-->>Dev: しきい値判定
    Dev->>Prod: promote
    Obs-->>Prod: 異常なら自動 rollback
```

Canonical test suite の最小構成

OpenAI は小規模でも targeted prompt set で回帰を早期検知できると案内しています。現実的には、Small でも最初から次の四群を分けると壊れにくくなります。 [31]

1. **Golden set**
   日次で通るべき代表 20–100 ケース。正常系の品質を見る。

2. **Breakage set**
   過去障害を再現するケース。更新時に最優先で回す。

3. **Adversarial set**
   prompt injection、曖昧入力、format 崩れ、tool misuse など。

4. **Replay set**
   本番ログの再生。stored completions / traces があるなら最も価値が高い。

移行チェックリスト

これは本レポートで最も実務価値が高い部分です。major update が出たら、次の順で進めるのが安全です。[32]

- 旧モデルの **snapshot / alias / current prompt version** を固定し、比較基準を凍結する
- **モデルだけ**新しくし、prompt はまだ触らない
- reasoning model なら **reasoning_effort** を pin する
- canonical / replay / adversarial の三種類で baseline eval を回す
- 失敗を **format / tool / safety / scope / long-context** に分類する
- prompt は **一回に一か所だけ**直す
- 形式崩れは prompt でなく **schema / tool / validator** 側で直す
- canary を 1–5% で流し、**p95 latency / schema validity / tool count / refusal rate** を監視する
- rollback 条件を超えたら **prompt version か model snapshot** を即時復元する
- 変更理由を changelog に残し、次回更新の再利用資産にする

ガバナンスと運用体制

NIST の GenAI Profile と AI RMF Playbook は、生成AIの運用を **Govern / Map / Measure / Manage** の継続的な活動として扱っています。Microsoft も監視・トレース・評価を AI ライフサイクル全体に組み込むべきだとし、OpenAI は tracing と evals、Anthropic は Analytics API と OpenTelemetry 対応を進めています。したがって、prompt 管理は個人技ではなく、**責任分界点のある運用機能**にすべきです。[33]

推奨ロール

ロール	主責務	主要成果物
Prompt Owner	prompt contract の品質責任	prompt spec、changelog、release note
Agent Engineer	orchestration / tool / fallback 実装	tool schema、router、runtime
Eval Owner	canonical / replay / adversarial の整備	eval dataset、grader、gate
Domain Owner	業務正しさの判定	acceptance criteria、例外ルール
Safety Owner	ガードレールと拒否境界の監督	safety eval、approval policy
SRE / Platform	観測・SLA・rollback	dashboards、alerts、canary、rollback runbook

推奨SLA

以下は私の推奨値であり、公式標準ではありません。ただし、ベンダー資料にある deprecation の常態化、継続評価、監視、rollback 能力を前提にすると、これくらいの運用速度が妥当です。[34]

- **P1 品質事故**: 30 分以内に rollback か traffic shift
- **major model update の一次評価**: 公開から 5 営業日以内
- **deprecation 公表後の代替候補比較**: 10 営業日以内に完了
- **定常監視レビュー**: 週次
- **prompt/eval 資産棚卸し**: 月次
- **本番ログ再生を含む回帰評価**: 主要変更ごと + 少なくとも週次

監視メトリクスの最小セット

- 成功率: task completion, final answer score
- 形式: schema validity, parser error rate
- 軌跡: tool count, trajectory match, retry count
- 安全: refusal rate, escalation rate, approval rejection rate
- 性能: p50/p95 latency, tokens per success, cache hit rate
- 事業: CSAT、一次解決率、CVR、担当者引き継ぎ率

これらは Google の final answer / trajectory、Microsoft の quality / latency / error / token 指標、OpenAI の trace grader、Anthropic の analytics/telemetry の方向性と整合します。[35]

ロールバック計画

もっとも重要なのは、**prompt rollback と model rollback を別に持つ**ことです。OpenAI は prompt の履歴から restore でき、OpenAI の model snapshots は挙動固定に使えます。AWS Bedrock は prompt version を static snapshot として扱います。したがって、変更単位を少なくとも「prompt version」「model snapshot」「tool schema version」の三軸に分けるべきです。[36]

実装上の runbook はシンプルで構いません。
Step A: 監視閾値超過を検知。
Step B: prompt 変更が原因か model 変更が原因かを切り分け。
Step C: prompt restore → 直らなければ model snapshot を戻す。
Step D: replay eval で再発防止ケースを追加。
この流れを自動化できるほど、次回以降の更新が安くなります。[37]

今後の見通し

私の見立てでは、**大きな prompt change が頻繁に必要な時期は、少なくとも今後 12–24 か月は続く**可能性が高いです。より慎重に言えば、**最前線APIでは 2027 年ごろまで、消費者向けサーフェスではそれ以降も**揺れが続く可能性があります。これは公式な約束ではなく、本レポートの推定です。ただし、根拠はかなりあります。OpenAI は GPT-4o の開始後 1 年で personality / helpfulness に関する mainline update を 5 回行ったと説明し、Model Spec 自体も継続更新される文書だとしています。Anthropic も system prompt が periodic に更新されると明言し、deprecations は routine で、歴史的に多くのモデルを初回公開から約 1 年で退役させてきたと述べています。[38]

なぜまだ続くかというと、各社がまだ **モデルの中身だけでなく、モデルの周辺インターフェース** を変え続けているからです。developer message への移行、structured outputs、tool calling、adaptive thinking、prompt registry、tracing、stateful responses、context compaction、app-side memory など、変化点が

「モデル本体」だけに留まっていません。提示された最新資料でも、GPT-5.5 向けには旧 prompt stack をそのまま持ち込むなとされ、OpenAI も Anthropic も agentic workloads 向けに model family ごとの guidance を別途用意しています。これは、まだ“普遍的な単一 prompt style”が成立していないことの裏返しです。[39]

ただし、**変化が完全に収まらない**としても、**更新耐性はかなり上げられる**はずです。安定化のシグナルとして見るべきなのは、次の五つです。

- ベンダーが **snapshot の保持期間**を長くし、alias 変更ではなく snapshot pinning を標準にする
- major release note における **prompt migration 項目**が短くなる
- critical path の大半が **schema / tool policy / validator** で固定され、自由文応答が周辺化する
- monitoring が **prompt 単位ではなく task contract 単位**になる
- deprecation 周期が年単位より長くなり、system prompt 更新頻度も下がる

これらが進むほど、「prompt をいじらないと動かない」領域は減り、「interface を保てば動く」領域が増えます。実際、OpenAI の reusable prompts / rollback / snapshots、AWS の prompt version snapshots、Google/Microsoft の structured output と evaluation、OpenAI/Microsoft の tracing は、その方向へ進んでいます。[40]

未解決点と限界もあります。第一に、ベンダーは挙動差分の理由を公開していても、**内部アーキテクチャの詳細**までは公開しないため、互換性破壊の原因を完全に因果分解することはできません。第二に、日本語の一次資料は Google / Microsoft / AWS / Anthropic の一部では豊富ですが、OpenAI の主要 release / migration 文書は英語中心で、その部分は英語の一次資料を使いました。第三に、将来予測は公開 cadence と現行方針からの推論であり、各社の将来計画を保証するものではありません。[41]

最終的な実務指針を一文で言えば、**プロンプトを“名人芸の文章”として管理するのをやめ、モデル更新に耐える“契約・評価・観測対象”として管理すること**です。そうすれば、GPT-5.5 や Claude 4.7 級の更新が来ても、修正対象はメガプロンプト全体ではなく、限定された layer に閉じ込められます。[42]

---

[1] [7] [9] [11] [12] [42] https://developers.openai.com/cookbook/examples/gpt4-1_prompting_guide
https://developers.openai.com/cookbook/examples/gpt4-1_prompting_guide

[2] [6] [23] [24] [25] [30] [36] [37] [40] https://developers.openai.com/api/docs/guides/prompting
https://developers.openai.com/api/docs/guides/prompting

[3] https://docs.anthropic.com/en/release-notes/system-prompts
https://docs.anthropic.com/en/release-notes/system-prompts

[4] [14] [41] https://model-spec.openai.com/2025-04-11.html
https://model-spec.openai.com/2025-04-11.html

[5] [39] https://developers.openai.com/api/docs/guides/prompt-guidance
https://developers.openai.com/api/docs/guides/prompt-guidance

[8] [19] [32] https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide
https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide

[10] https://docs.anthropic.com/ja/resources/prompt-library/data-organizer
https://docs.anthropic.com/ja/resources/prompt-library/data-organizer

[13] https://developers.openai.com/api/docs/guides/direct-preference-optimization
https://developers.openai.com/api/docs/guides/direct-preference-optimization

15 https://docs.anthropic.com/en/docs/about-claude/pricing
https://docs.anthropic.com/en/docs/about-claude/pricing

16 26 28 https://developers.openai.com/api/docs/guides/structured-outputs
https://developers.openai.com/api/docs/guides/structured-outputs

17 38 https://openai.com/index/expanding-on-sycophancy/
https://openai.com/index/expanding-on-sycophancy/

18 https://openai.com/index/function-calling-and-other-api-updates/
https://openai.com/index/function-calling-and-other-api-updates/

20 21 35 https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/evaluation-agents?hl=ja
https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/evaluation-agents?hl=ja

22 https://openai.com/api/pricing/
https://openai.com/api/pricing/

27 https://developers.openai.com/tracks/building-agents
https://developers.openai.com/tracks/building-agents

29 https://developers.openai.com/api/docs/guides/reasoning-best-practices
https://developers.openai.com/api/docs/guides/reasoning-best-practices

31 https://developers.openai.com/blog/eval-skills
https://developers.openai.com/blog/eval-skills

33 https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence
https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence

34 https://support.claude.com/en/articles/12738598-adapting-to-new-model-personas-after-deprecations
https://support.claude.com/en/articles/12738598-adapting-to-new-model-personas-after-deprecations