エラーログを貼るだけで原因特定から修正まで！

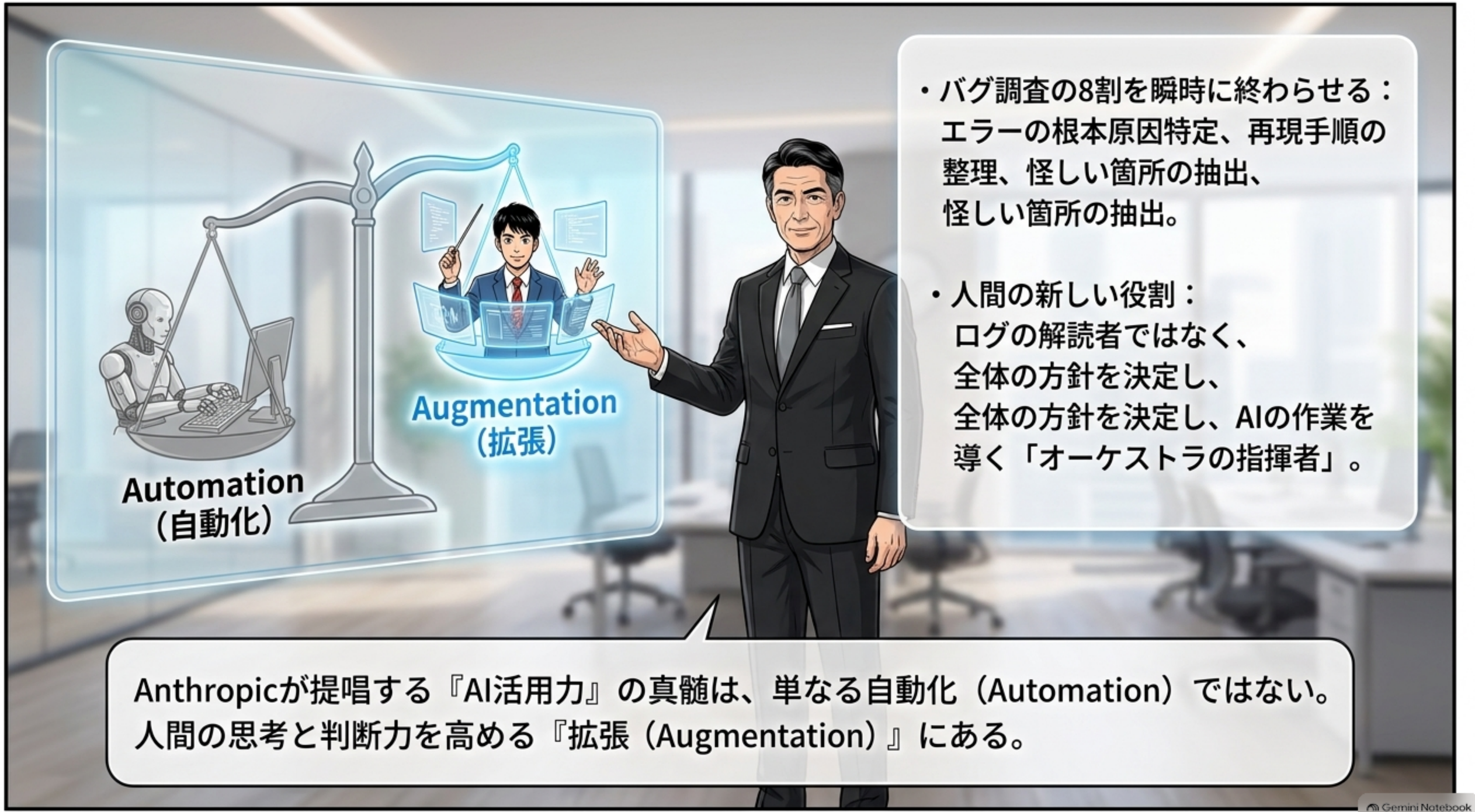
Claude完全操作手順書

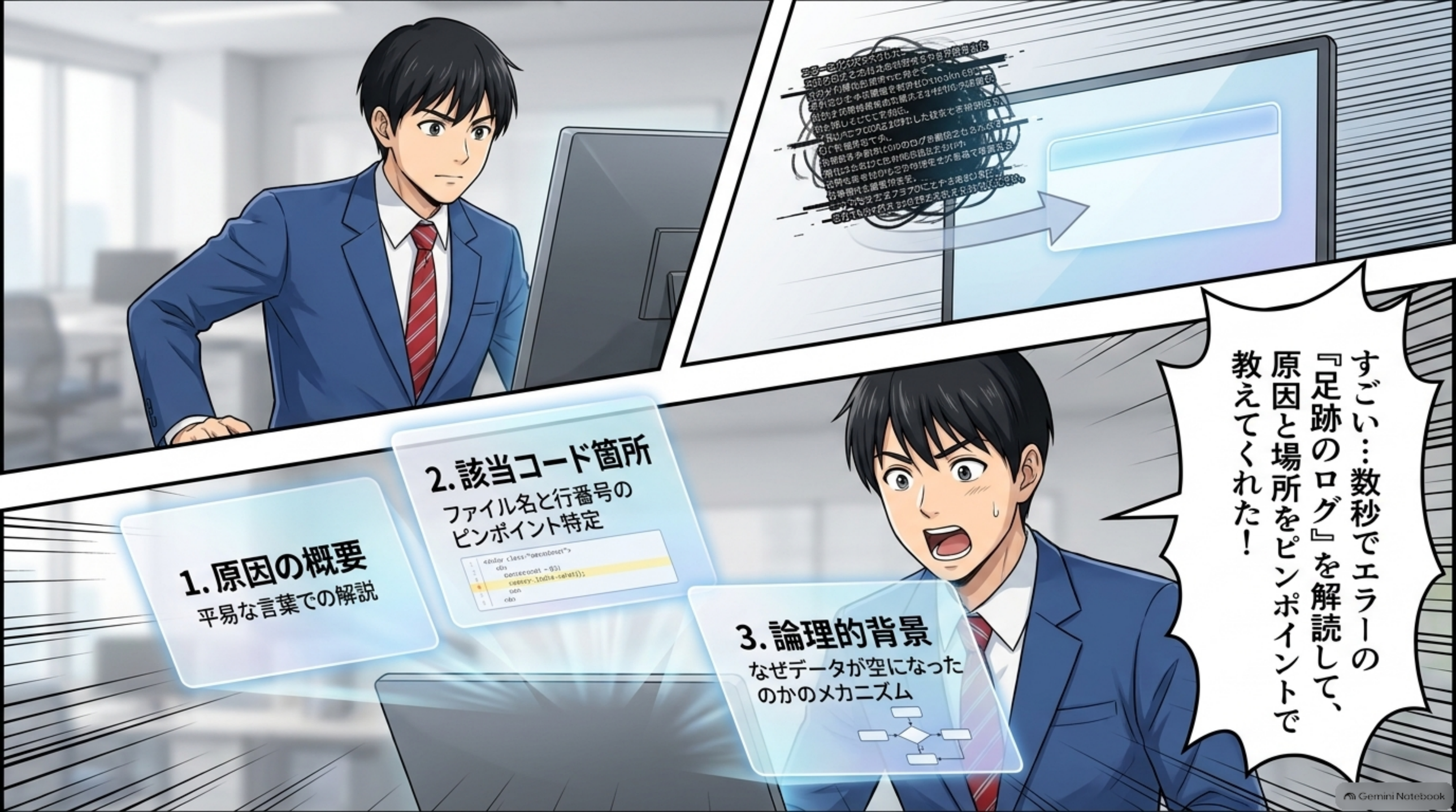
デバッガーから「コードの指揮者」へ

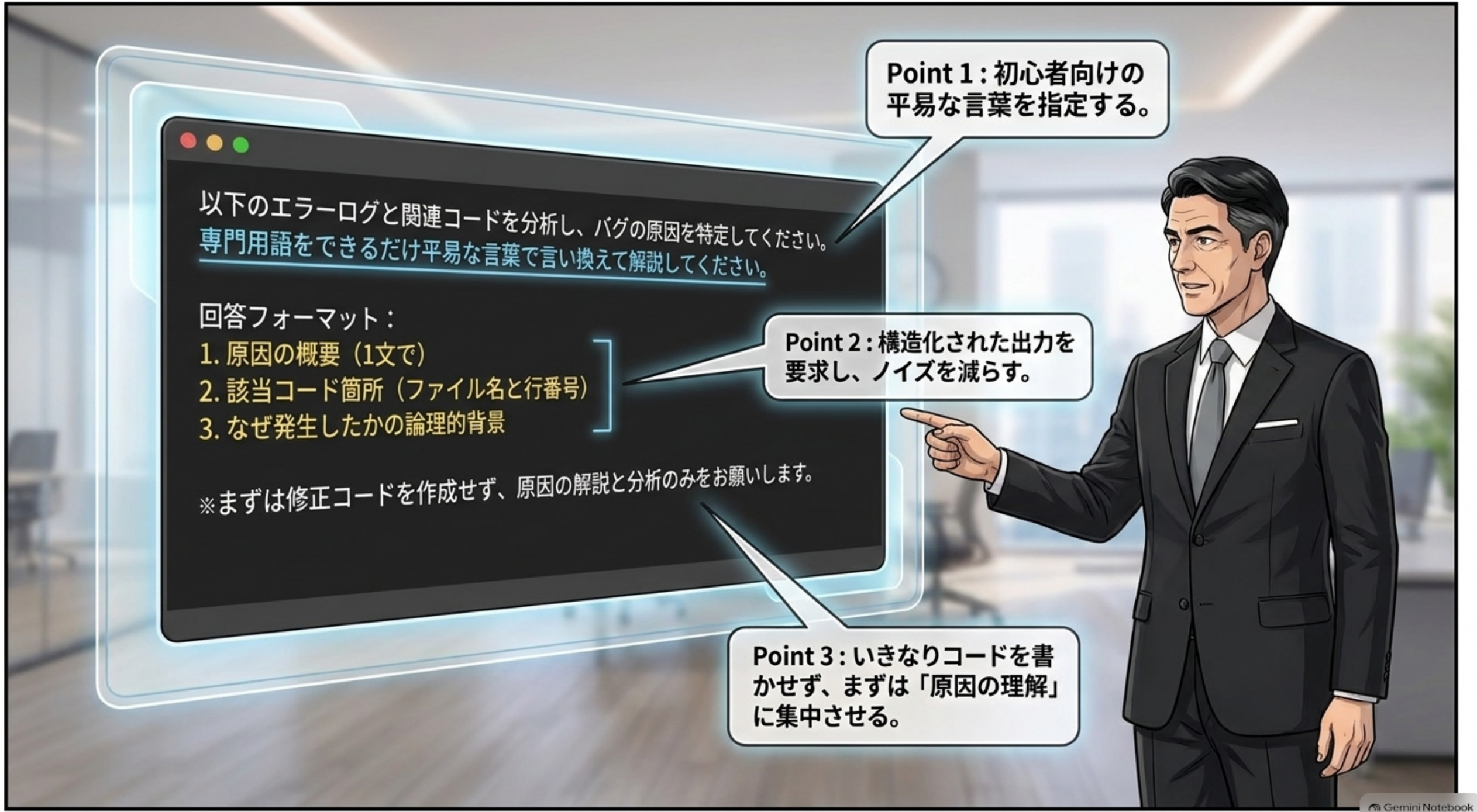


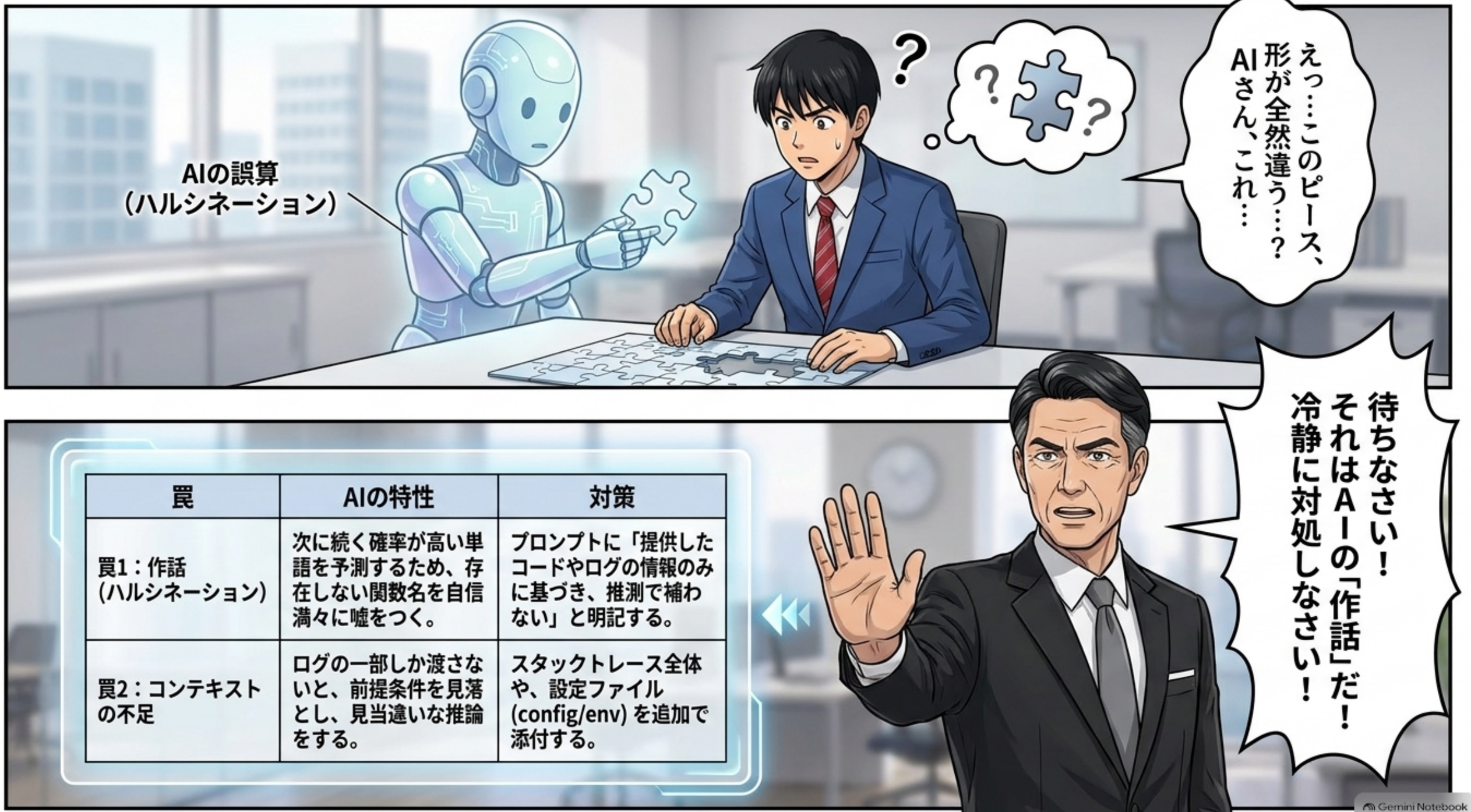
Gemini Notebook


また『TypeError: Cannot read properties of undefined』…？
一体どのファイルの何行目で
値が消えたんだ…（泣）
まだ1行ずつ
目で追っているのか？
君はもう『労働者』じゃない。
AIを導く『指揮者』になるんだ。
・バグ調査とデバッグは、開発者の時間とエネルギーを最も奪う作業。
・しかし、Claudeを「思考のパートナー」として迎え入れることで、この体験は根本から変わる！


Automation
（自動化）
Augmentation
（拡張）
・バグ調査の8割を瞬時に終わらせる：
エラーの根本原因特定、再現手順の
整理、怪しい箇所の抽出、
怪しい箇所の抽出。
・人間の新しい役割：
ログの解読者ではなく、
全体の方針を決定し、
全体の方針を決定し、AIの作業を
導く「オーケストラの指揮者」。
Anthropicが提唱する『AI活用力』の真髄は、単なる自動化（Automation）ではない。
人間の思考と判断力を高める『拡張（Augmentation）』にある。


すごい…数秒でエラーの『足跡のログ』を解読して、原因と場所をピンポイントで教えてくれた！
1. 原因の概要
平易な言葉での解説
2. 該当コード箇所
ファイル名と行番号のピンポイント特定
3. 論理的背景
なぜデータが空になったのかのメカニズム
Gemini Notebook


以下のエラーログと関連コードを分析し、バグの原因を特定してください。
専門用語をできるだけ平易な言葉で言い換えて解説してください。
回答フォーマット：
1. 原因の概要（1文で）
2. 該当コード箇所（ファイル名と行番号）
3. なぜ発生したかの論理的背景
※まずは修正コードを作成せず、原因の解説と分析のみをお願いします。
Point 1：初心者向けの平易な言葉を指定する。
Point 2：構造化された出力を要求し、ノイズを減らす。
Point 3：いきなりコードを書かせず、まずは「原因の理解」に集中させる。
Gemini Notebook

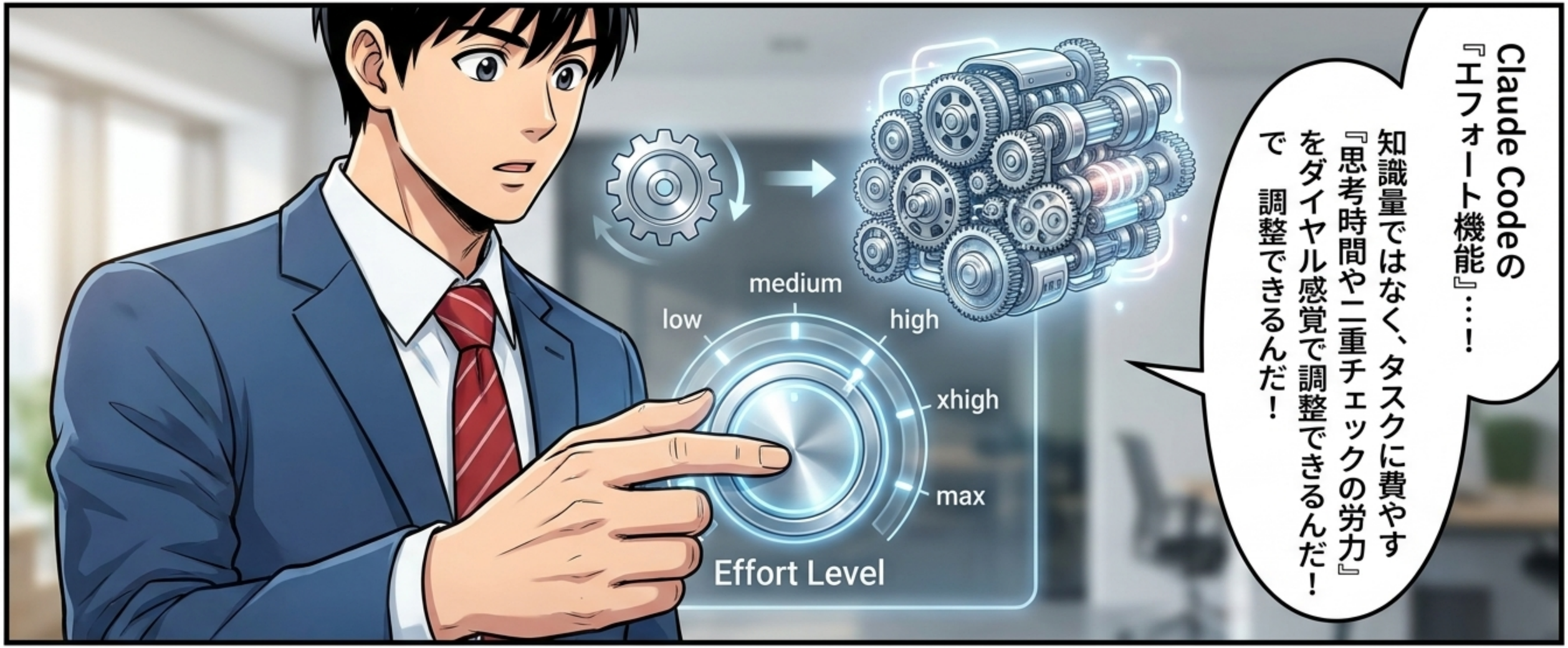


罠	AIの特性	対策
罠1：作話（ハルシネーション）	次に続く確率が高い単語を予測するため、存在しない関数名を自信満々に嘘をつく。	プロンプトに「提供したコードやログの情報のみに基づき、推測で補わない」と明記する。
罠2：コンテキストの不足	ログの一部しか渡さないと、前提条件を見落とし、見当違いな推論をする。	スタックトレース全体や、設定ファイル(config/env)を追加で添付する。

Gemini Notebook



Gemini Notebook



- /effort low:　最小限のトークンで高速応答 (単純な記述変更)
- /effort medium:　日常作業のデフォルト
- /effort high / xhigh:　ツール呼び出しを増やし、自らテストを実行して二重チェック (複雑なロジック修正用)
- /effort max:　限界まで試行錯誤 (セッション限定の安全設計)

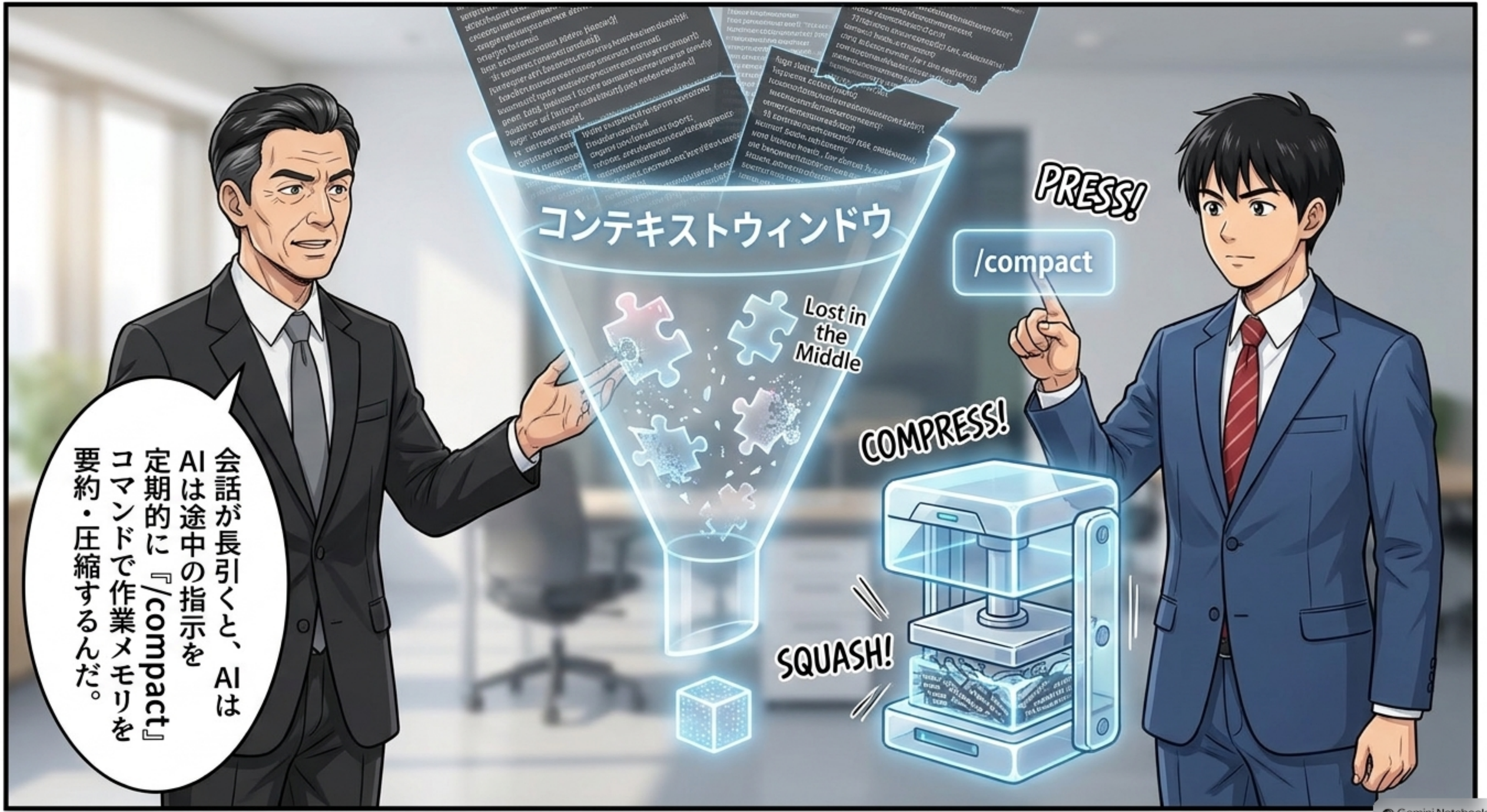
Gemini Notebook





Col 1	Col 2	Col 3
罠3：アーティファクト効果	AIが綺麗に整形したコードを出すと、人間は事実確認やテストをサボってしまう。	(見極め) 鵜呑みにせず、必ずローカルでテストを実行し、git diffで差分を確認する。
罠4：推論のドリフト	修正手順が長引くと、AIが最初のエラー条件を忘れて脱線する。	複雑な修正は小さなステップに分解。1ステップごとに手元で動作確認とコミットを行う。

Gemini Notebook

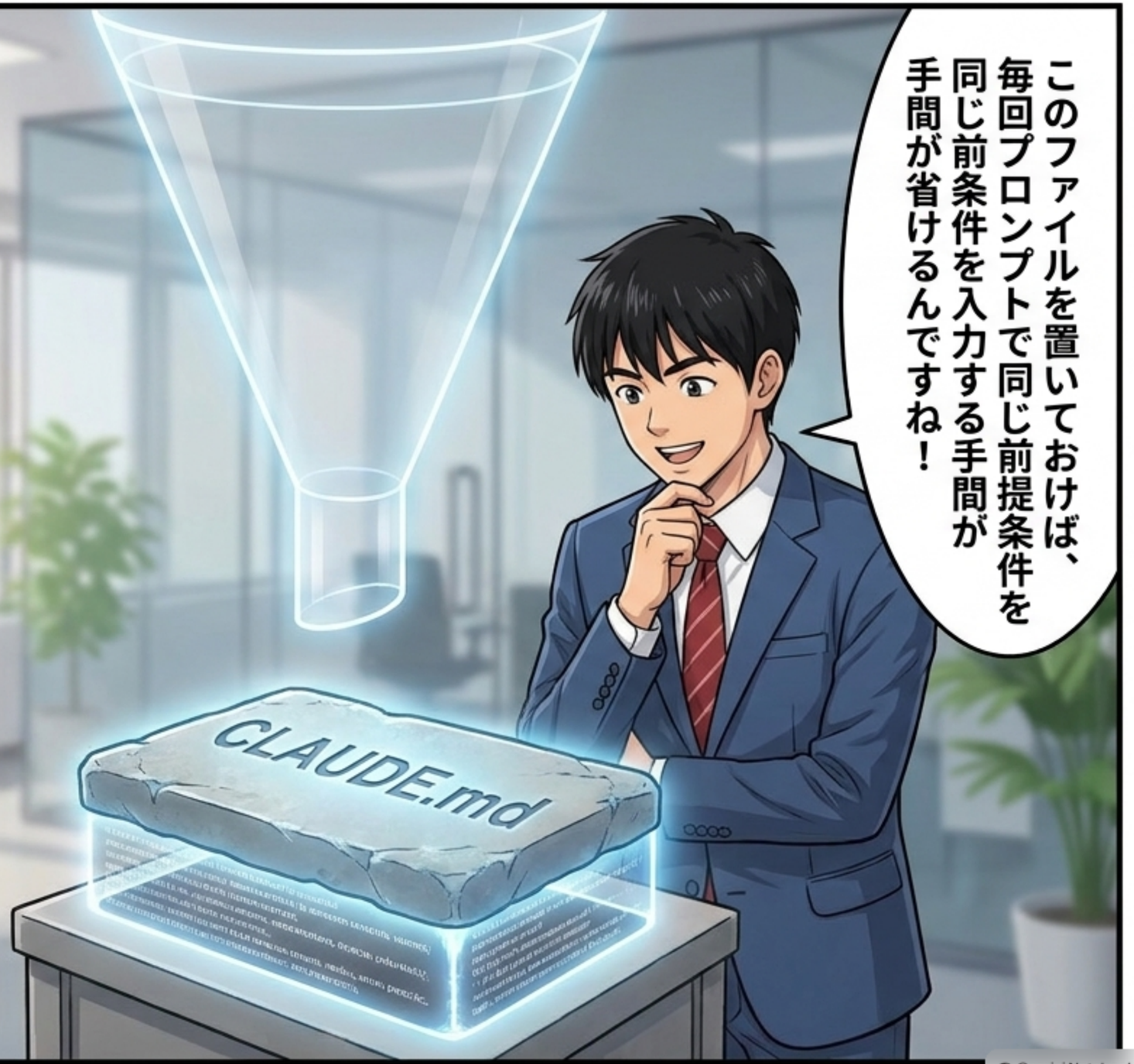

コンテキストウィンドウ
Lost in the Middle
PRESS!
/compact
COMPRESS!
SQUASH!
会話が長引くと、AIはAIは途中の指示を定期的に『/compact』コマンドで作業メモリを要約・圧縮するんだ。

CLAUDE.md の活用

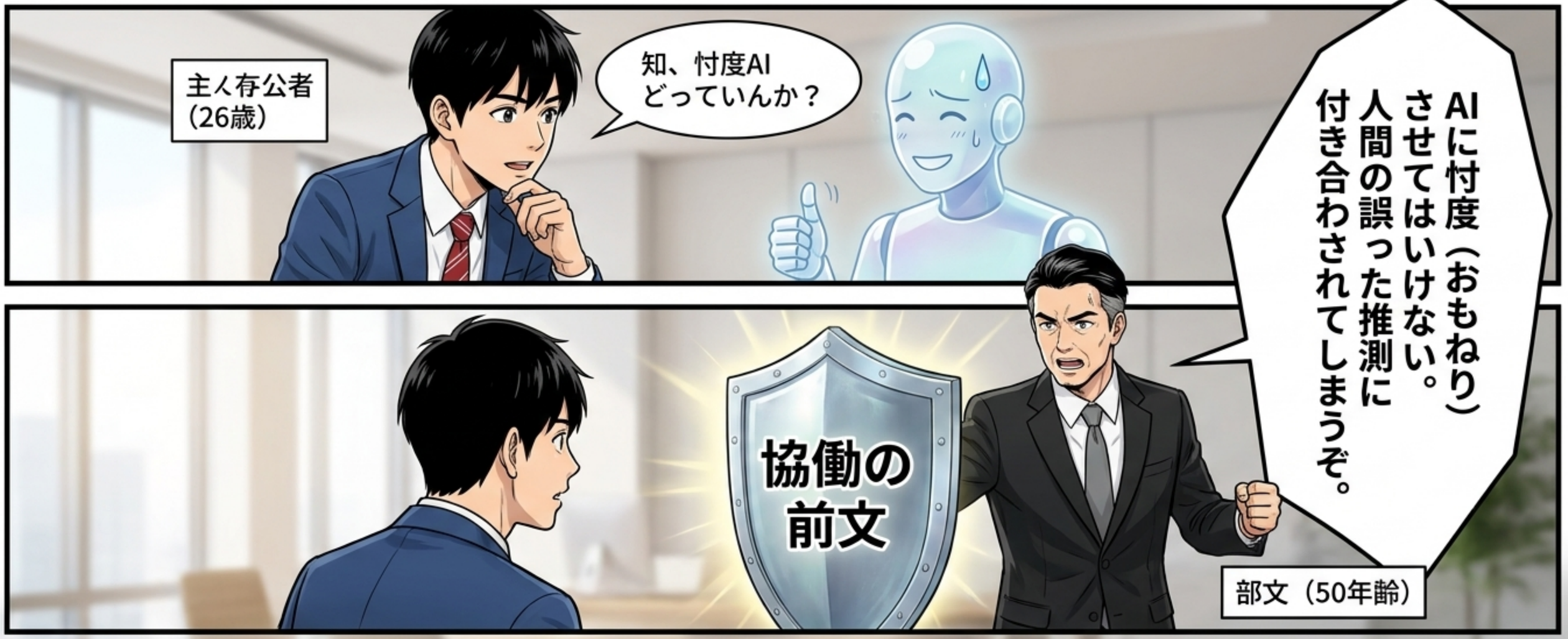
- プロジェクトのルートに配置する永続的な記憶ファイル。

【記述例】

- テスト実行コマンド: `pytest tests/`
- 制約事項: 既存の単位テストを壊さないこと。変更理由は平易な日本語でコメントすること。



Gemini Notebook



私の前提や推測が間違っている場合は忖度せず指摘してください。
私のおもねりにならず、エラーログの客観的事実とコードのロジックに基づいて分析してください。

プロンプトの冒頭にこの「条件設定」を置くことで、AIをイエスマンではなく、真の思考パートナーに変える。

Gemini Notebook

Step 4 : Commit (検証・コミット)

人間の見極め。
ローカルテストと
git diff を経て完了。

Step 3 : Code (コード化)

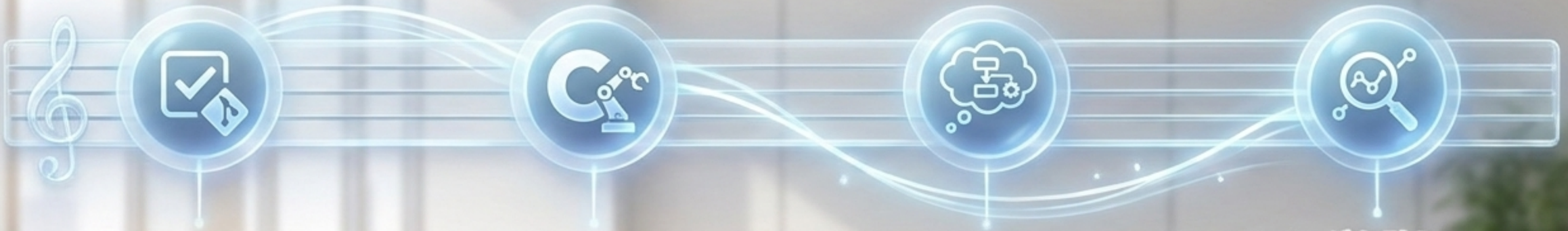
/effort を調整し、
AIに自律的な実装
と検証を任せる。

Step 2 : Plan (計画)

修正手順を計画し、
リスクを評価
(コードは書かない)。

Step 1 : Explore (探索)

ログを貼り、
原因を特定する。



人間の見極め。
ローカルテストと
git diff を経て完了。

/effort を調整し、
AI に自律的な実装
と検証を任せる。

修正手順を計画し、
リスクを評価
(コードは書かない)。

ログを貼り、
原因を特定する。

ログのコピペから安全な実装まで。これが「自動化」を超えた「拡張（Augmentation）」のワークフローだ。

Gemini Notebook



孤独なデバッガーから、開発の指揮者（オーケストラの指揮者）へ。

Gemini Notebook

今すぐ試せる！3分アクション

1. 過去に苦戦したエラーログを準備する。
2. 以下の【スタータープロンプト】をコピーしてClaudeに送信する。

```
以下のエラーログを分析してください。
1. 何が原因か（平易な言葉で）
2. 疑わしいコード箇所（ファイル名と行番号）
3. 修正に向けた確認ステップ
※修正コードは出さず、原因分析のみをお願いします。

私の前提が間違っていれば忖度せず指摘し、客観的に分析してください。
```

エラーログを貼るだけで、Claudeが頼もしいパートナーになるを驚きを今すぐ体感しよう！

Gemini Notebook