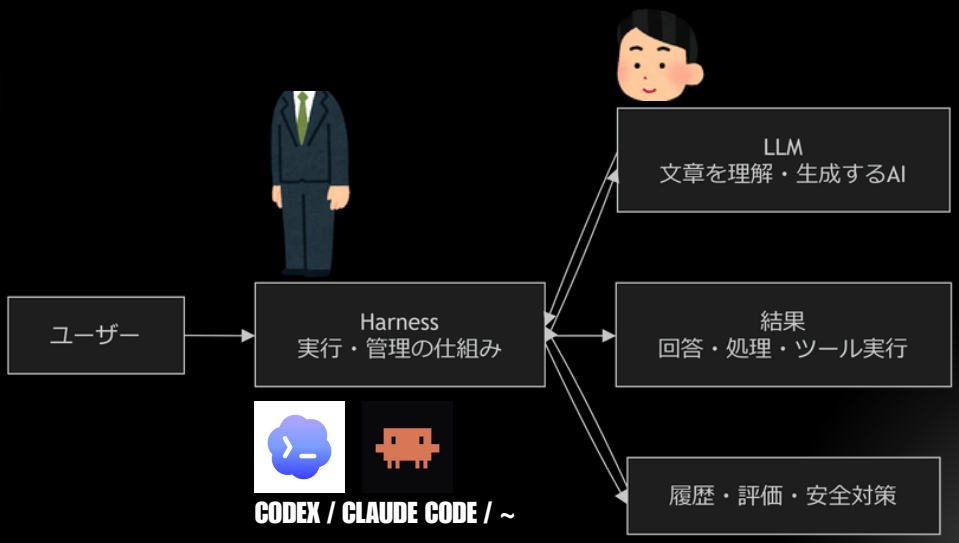
これから「AIエージェントのハーネス」の話をさせていただきます。
よろしくお願いします。

流れは大きく分けて3つになってます。
ハーネスとはなにかを軽く話して、効果と紹介をしていきます。

まずはハーネスとはなにかです。

ハーネスとは



既にAIエージェントを触ったことがある方ならわかると思いますが有名なCodexやClaude Code、あれ自体がハーネスです。
簡単にいうと中のAIに案内したり制御したりしてる感じです。
簡単に例えるとAIが頭でハーネスが身体みたいな感じです。

次にハーネスの効果です。

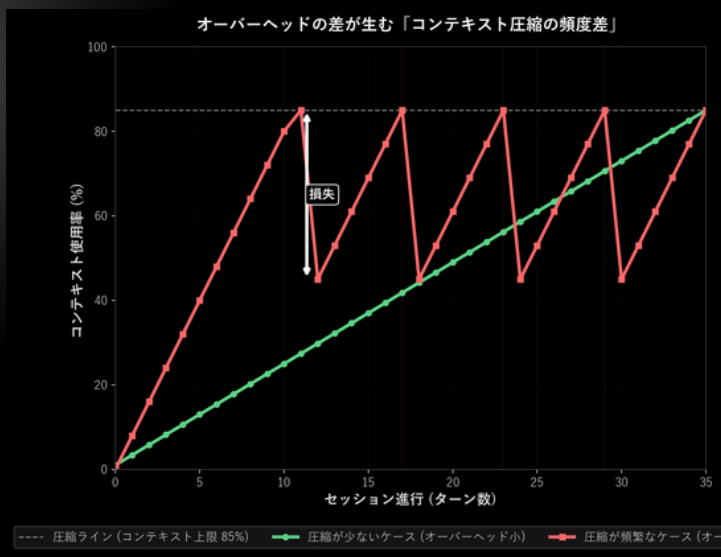
効果① コスト

- 固定オーバーヘッド
  「システムプロンプト + ツール定義」
  新規セッション/キャッシュ無効 → コスト 高
  固定オーバーヘッド 大 → コスト 高

- キャッシュヒット率
  ヒット率 高 → 再計算が減る → 入力コスト 低
  ヒット率 低 → 再計算が多い → 入力コスト 高

まずエージェントの使用コストが変わります。
特に大きく関係するのが固定オーバーヘッドとキャッシュヒット率です。
固定オーバーヘッドっていうのはAIに教える決まり事です。
この決まり事のサイズがハーネスによって結構違うので確認しとくといいと思います。
次にキャッシュヒット率です。
AIには毎ターン全文脈がAIに渡されますが、前回と共通する部分はすでに計算済みのため、計算結果を再利用できます。この「使い回せる範囲」が広いほど、キャッシュヒット率が高くなり、計算コストも下がります。
ハーネスによってコンテキストの扱いが異なるのでハーネスを選ぶ際はキャッシュヒット率も確認しておくといいと思います。

効果② 性能

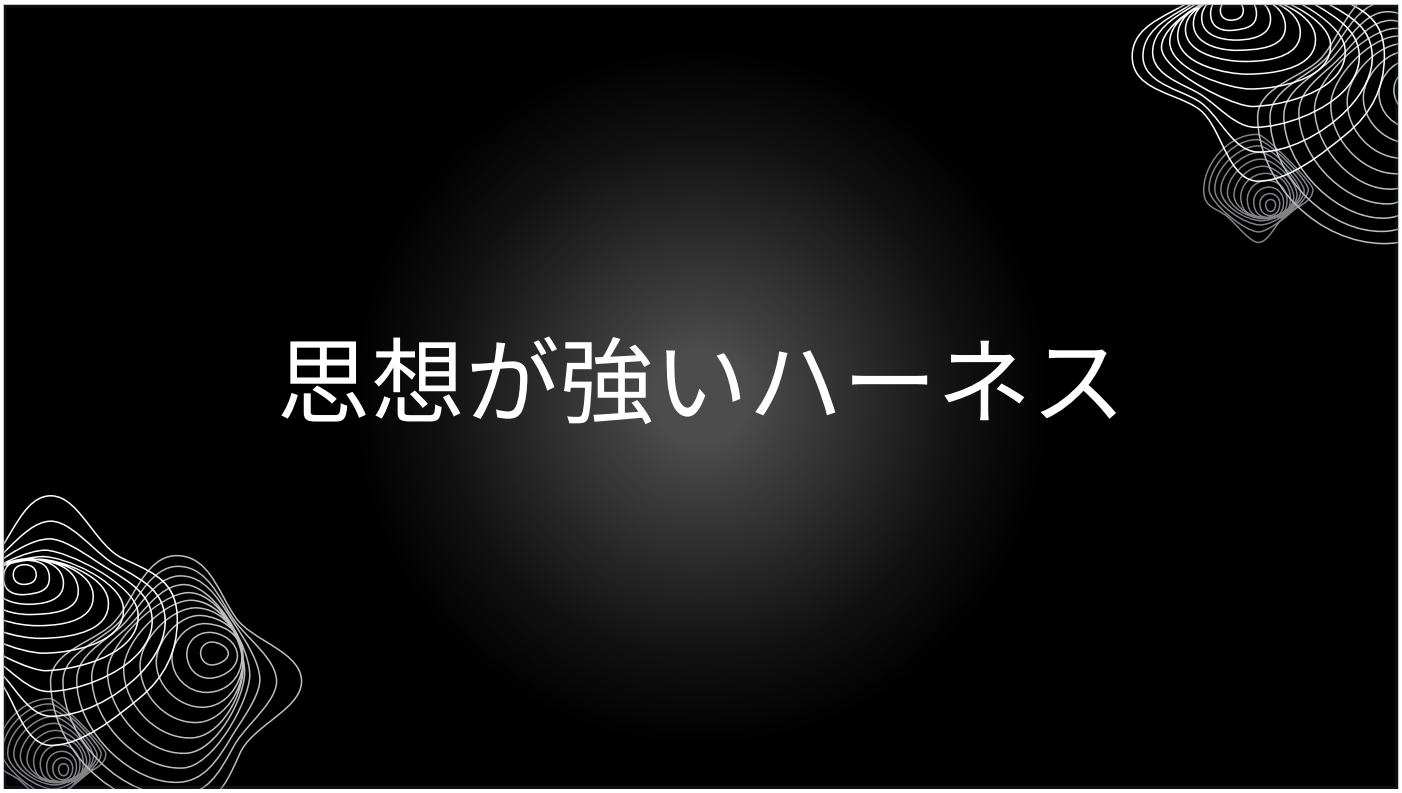


圧縮時に一部コンテキストが
欠落してしまう(≒記憶喪失)
圧縮はなるべく避けたい
→低オーバーヘッドが有利

そして性能も変わってきます。
無駄なプロンプトや出力を抱えるとコンテキストが圧迫され圧縮する必要が出てきます。
圧縮はコンテキストをAIが要約して短くすることです。
この際に後々必要になる細部の内容(変数名、過去の決定理由など)が切り捨てられることがあるためミスを誘発します。
ミスの根源であるこの圧縮を減らしたければ無駄なコンテキストを積まない、書き換えを最小限にするハーネスを採用するのがいいと思います。

ハーネスの紹介です。全部紹介してたらキリがないので僕が軽く知ってるやつだけ紹介します。

まず王道のハーネスですね。AIエージェントを初めて使う人はまずこちらのどちらかを使うといいと思います。
一般的な機能を持っていて動作も安定しているためどちらもおすすめです。
どっちを選ぶかは使うAIのモデルに合わせればいいと思います。

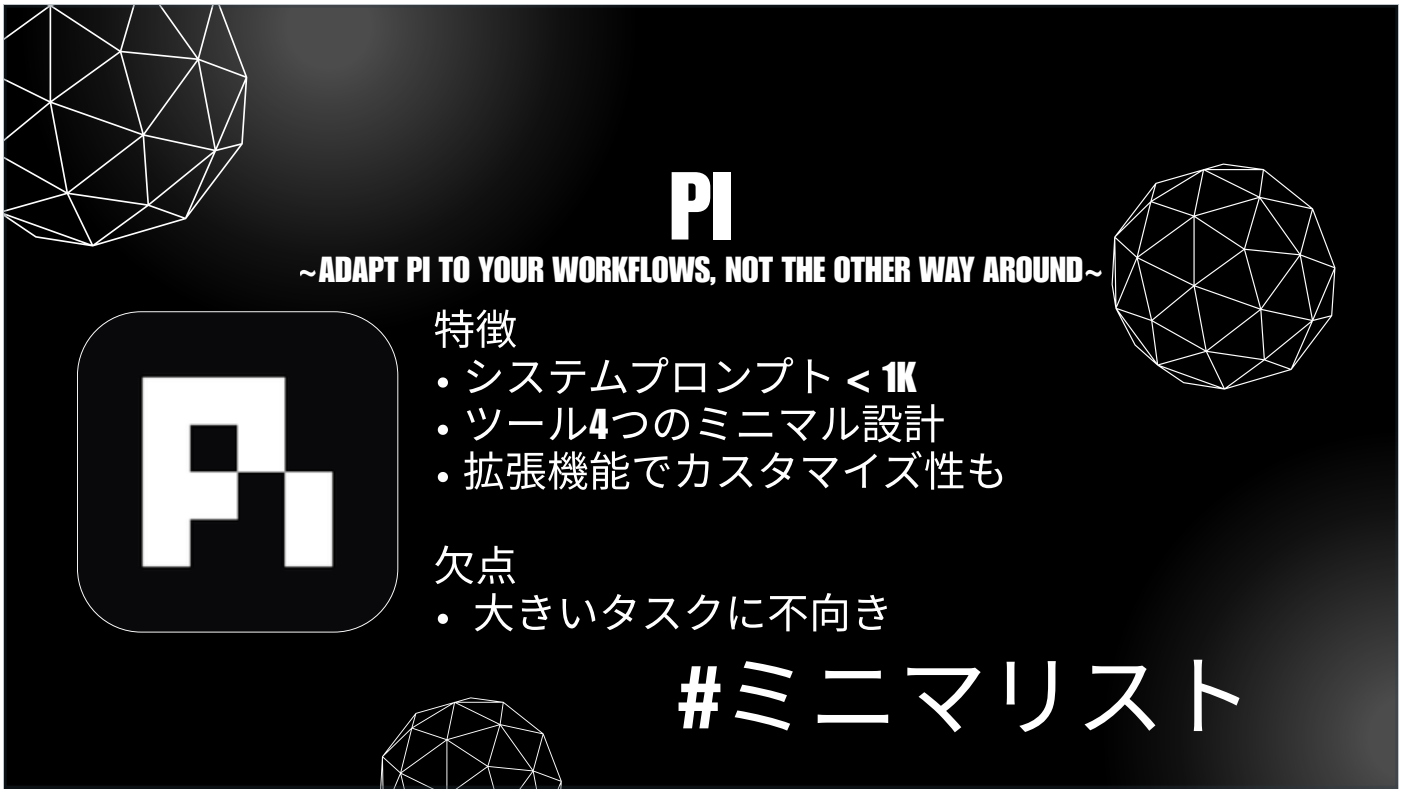


次に思想が強いハーネスです。

この三つですね。
左からdeepseek harness, pi, oh my piです。
普通のハーネスじゃ落ち着かない人におすすめです。
一つずつ軽く説明します。

まず、deepseek harness。
everything is a plugin.「すべてがプラグイン」という思想を持ってます。
こちらはプラグインベースで出来てるので、骨格からプラグインで構成されてます。
エージェントループなど深いとこまでプラグインで変更できるようになっています。
またCLIからプラグインの管理ができるのでエージェントがプラグインを扱うことができて魅力的です。
AIが普及したこの時代にマッチした思想だと思います。

ただプラグインに依存してるため壊れてるプラグインが一つでもあると起動できません。
また現在プレビュー版のため破壊的変更があり、アプデでプラグインが壊される可能性が高いです。
今後に期待ですね。



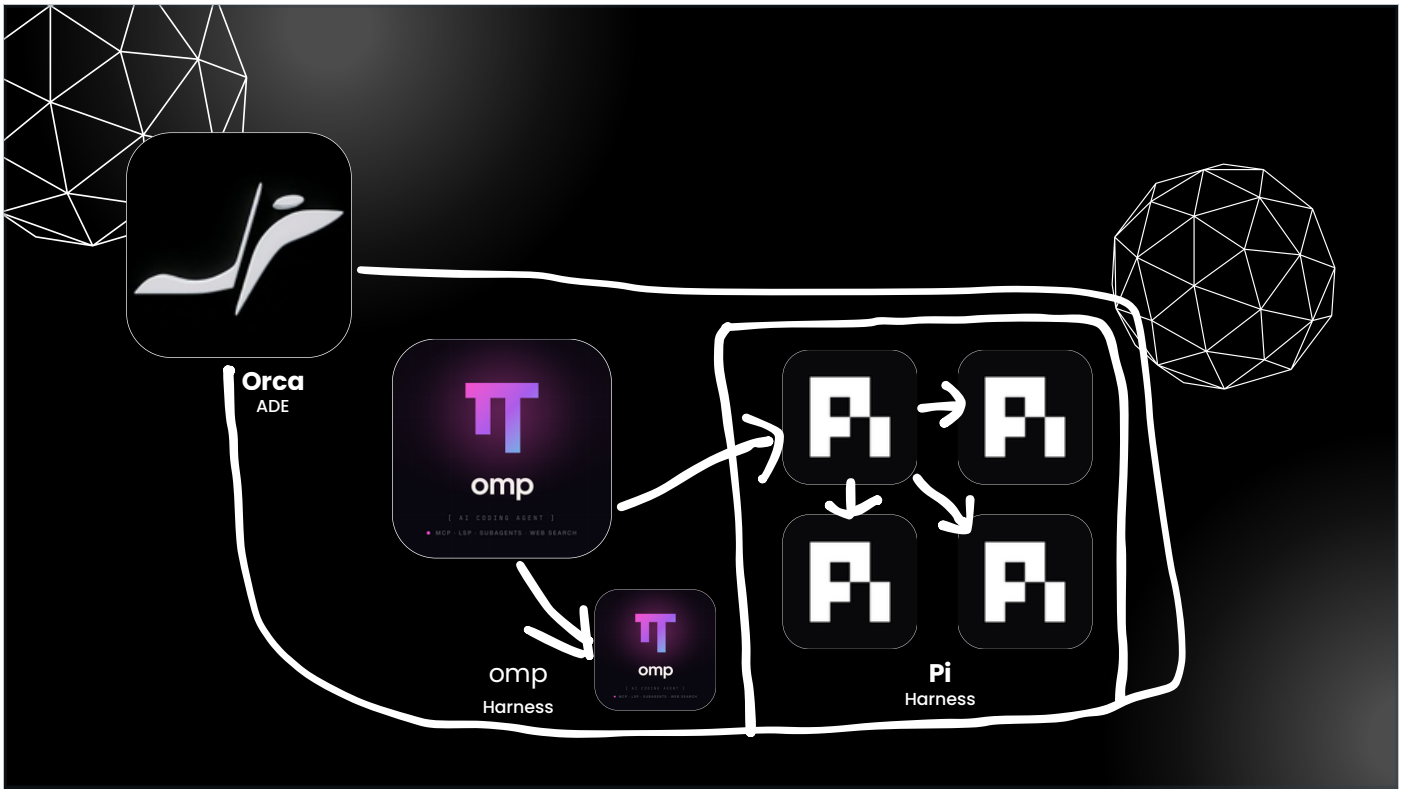
次、piです。
思想は「ワークフローをPiに合わせるのではなく、Piをワークフローに合わせる」とちょっとややこしいことを言ってます。
システムプロンプトがとにかく短く、ツールは読み込み、書き込み、編集、bashの4つのみです。
必要な機能は拡張機能で追加していくスタイルでミニマリストにピッタリなハーネスです。
実際使っているとキャッシュヒット率が常に99%~100%で推移していたので低オーバーヘッドと高いキャッシュヒット率からコストを抑えたい方にとてもおすすめです。
またメモリ使用率も軽いので複数のエージェントを並列で使う方にも向いてると思います。

最後にohmypiです。
IDEを配線したコーディングエージェントといわれています。
piのフォークから作られてますが最小限のpiとは真逆の思想を持っています。
まずコア部分がTypeScriptからRustに書き換えられています。
そしてPiが削ったサブエージェントやLSP(Language Server Protocol), DAP(Debug Adapter Protocol)などいろいろ入っててコーディングからデバッグまでできるようになってます。

ですが機能が多すぎて学習コストが高いだとかTUIが重いという評価もあります。
また機能が多いのでシステムプロンプトも大きくなってしまってるみたいです。

実際使ってみた感じは結構よかったです。1台のエージェントで開発する際はこれでいいと思いました。

ここで僕が考えた最強の構成を紹介します



これですね
Orcaという並列にエージェントを動かせるADEの中でOMPを使います。基本的なタスクはompで潰します。
タスクが多い場合はPiのオーケストレーションに投げてompは設計を守りきるという構造にします。
また連続したタスクがある場合は成果物の確認作業をもう一体のompに投げます。こうすればメインのompは設計やタスクを保持したまま作業を進めていけるという考えです。
構成に正解とかはないですが用途に合わせてハーネスを選ぶのがいいと思います。

最後にまとめです。

AIエージェントのコストや性能はモデルだけでなくハーネスでも改善することができます。
ハーネスはたくさんあり、各々いろんな思想を抱えてます。
そして合うハーネスは人それぞれです。
自分の好きな開発手法に合わせてメインのハーネスを持って置きながらも、目的や用途、環境に合わせて使い分けるのがベストかなと思います。

以上です。ご清聴ありがとうございました。