About My Coworkers

When AI Joined My Personal Development Team

A candid look at how artificial intelligence is transforming the developer experience.
And why you might be missing out.

About My Coworkers

When AI Joined My Personal Development Team

A candid look at how artificial intelligence is transforming the developer experience and why you might be missing out.

Thank You Sponsors!

Nick Nisi

- Developer Experience at **WorkOS**
- Organizer at **NebraskaJS**
- Podcaster
  - formerly of **JS Party**, now **Dysfunctional**
  - **Software Engineering Daily**
- Emcee at **SquiggleConf** next month
- AI Engineer?

Find me at **vim.dad**

If you're not experimenting

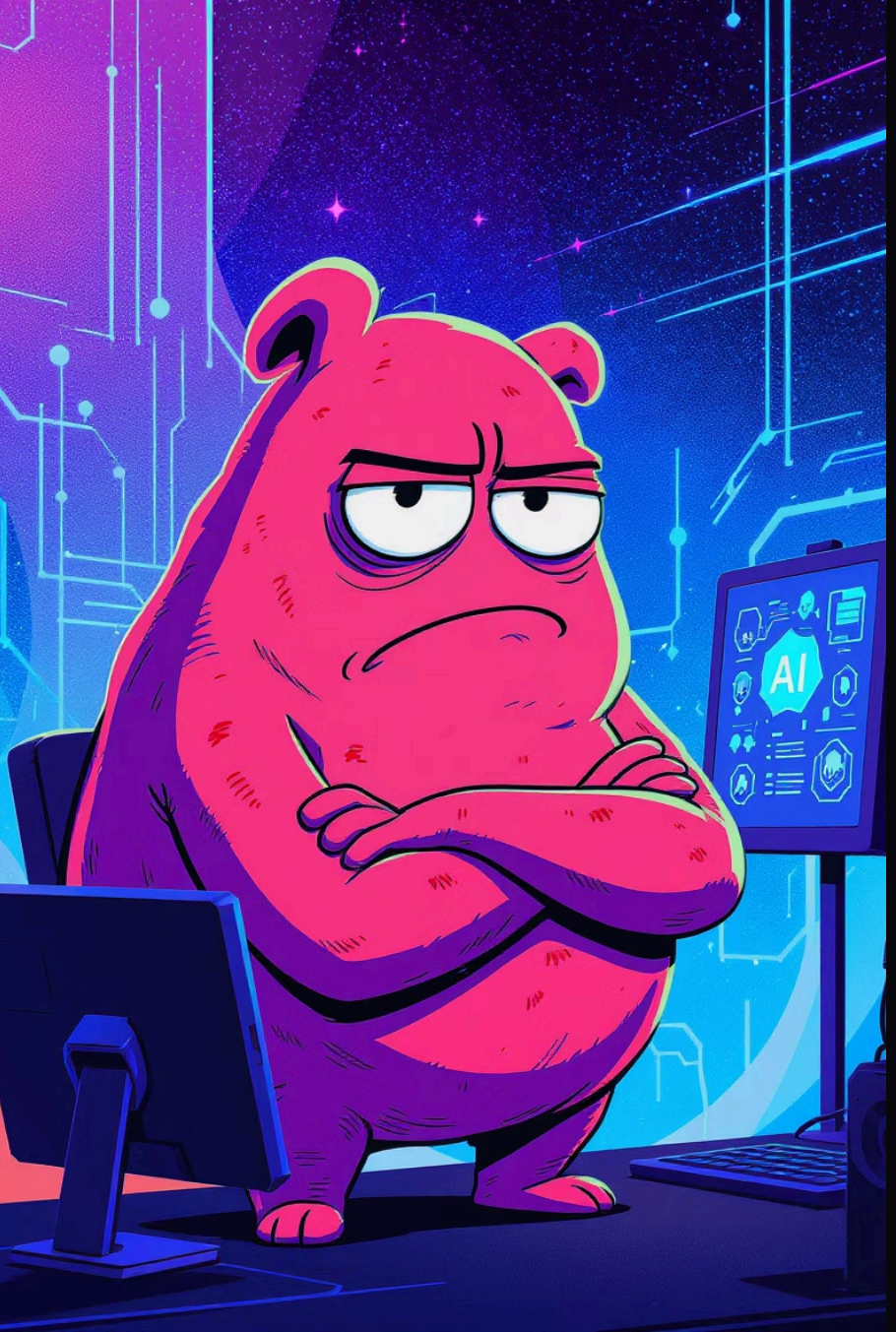
with **AI**

TODAY

You're

ALREADY BEHIND.

(That was meant to be provocative)



I also used to be **skeptical**

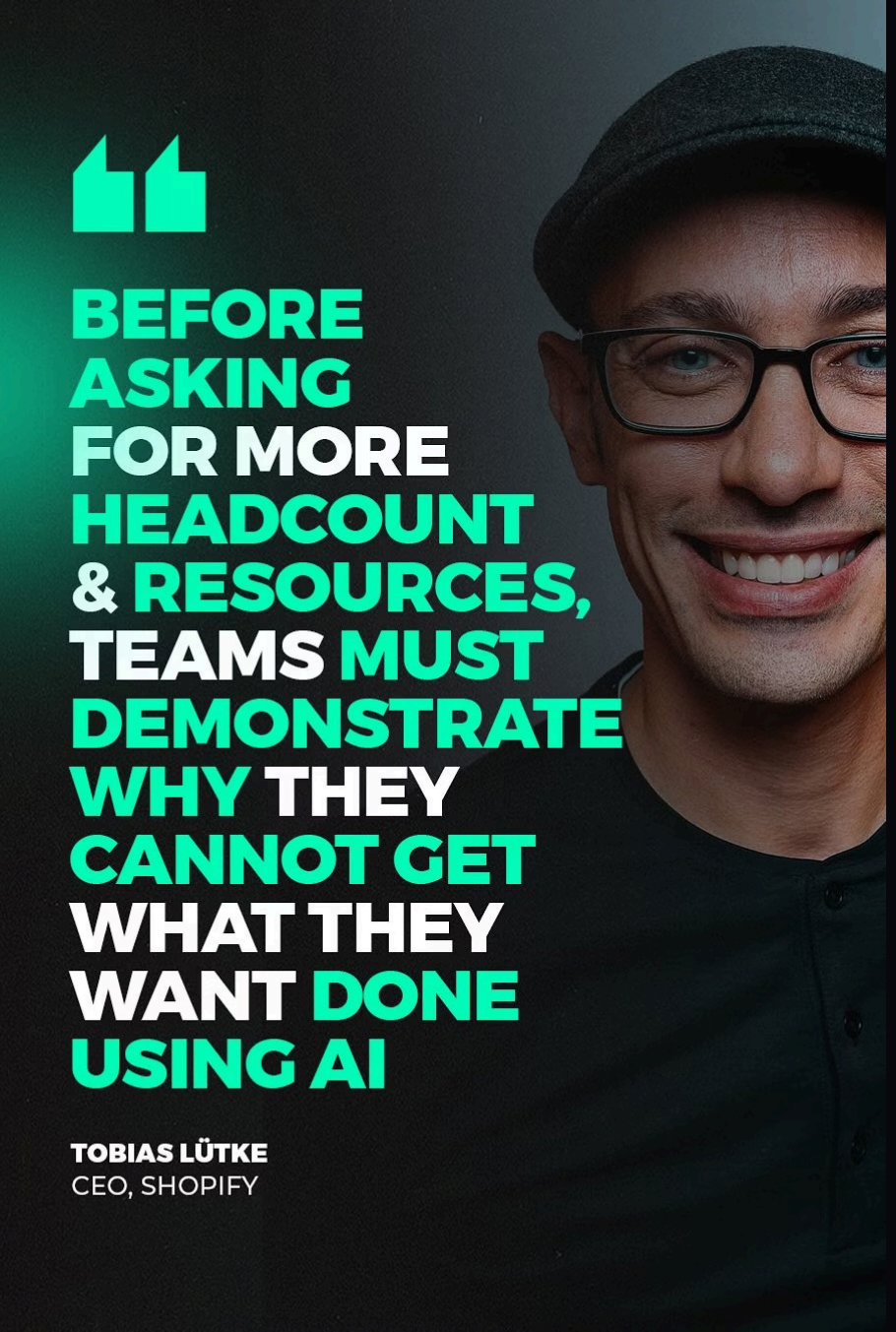
It just wasn't very good.

- Experimenting with Copilot
- Occasionally asking ChatGPT a question
- Whatever Meta had
- Felt like using it was "cheating"

JS Nation / React Summit

- These amazing developers weren't hiding anything
- They were **collaborating** with AI
- While I was hiding, they were innovating

That was my wakeup call.



I'm not the only one...

- AI mandate at Shopify
- Other companies too

⚠ The writing is on the wall – **AI adoption isn't optional anymore.**

Are we overreacting?

Maybe.

"If AI can solve your problems, you don't have hard problems."

What they're really saying is

"AI didn't work for me"

- Was their problem really that hard? **Maybe.**
- Were they just not good at structuring the problem for AI? **Probably.**
- Does this mean AI is bad and will never be good? **NOPE.**

This is the worst it'll ever be

We still need to learn how to engage it and when not to use it.

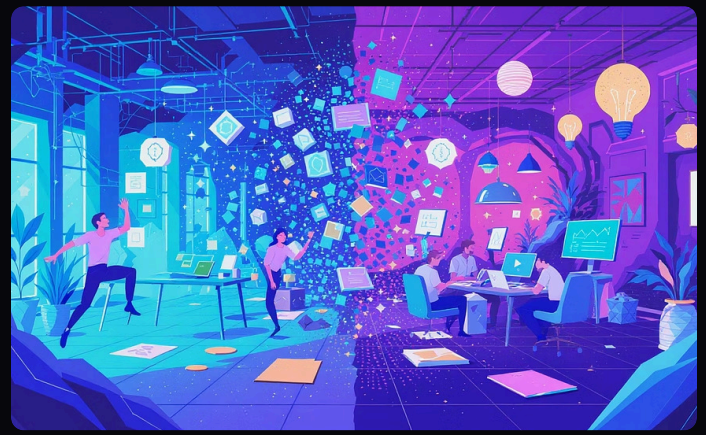
The AI Engineer Mindset

- The tools come and go, but the concepts remain
- AI isn't ment to **replace** you, it's meant to **augment** you.
- Your responsibility definitely hasn't increased. But it has **evolved**.

AI Likely won't take your job

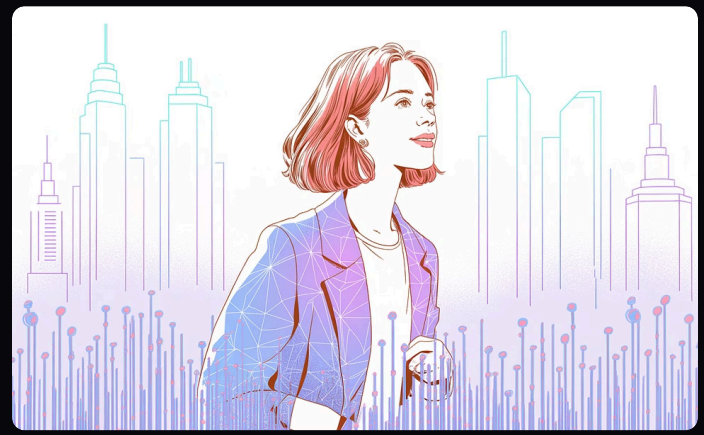
Someone using AI might though

The real competition isn't AI, it's those who embrace and leverage it effectively.



Ready or not, everything's about to change

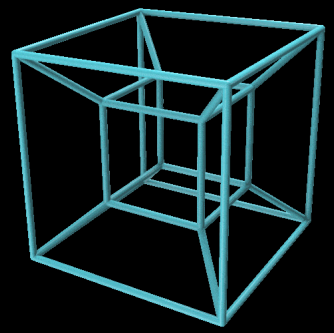
Your daily workflow is about to look completely different. Are you prepared to adapt?



That spark you felt when you first coded? Will it survive?

Prepare to rethink what your ideal job entails as AI reshapes daily tasks and responsibilities.

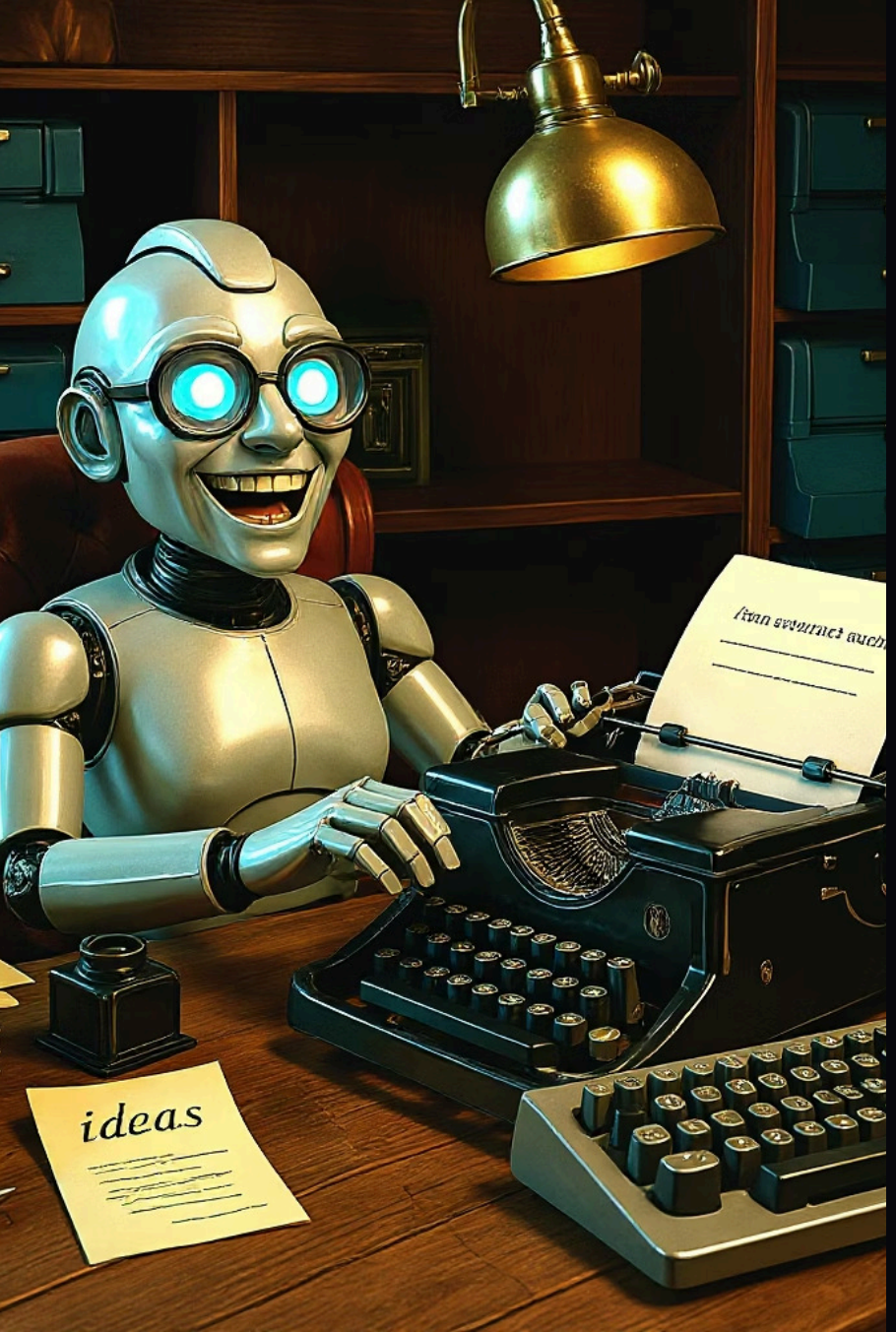
The dimensions of AI Adoption



1st Dimension: Smarter Autocomplete

- GitHub Copilot suggests the next line
- You hit tab to accept
- Still basically autocomplete, just smarter

This autocomplete is smart - I can kind of chat with it via comments.



2nd Dimension: Having Conversations

- Copy code into ChatGPT
- Ask contextual questions about the code

I can chat with this - Maybe I can paste in my file

It's manual.

It's tedious.

But it works.

The "copy-paste-chat" method, while effective, still requires significant human effort and can be repetitive. It's a stepping stone to more integrated AI workflows.

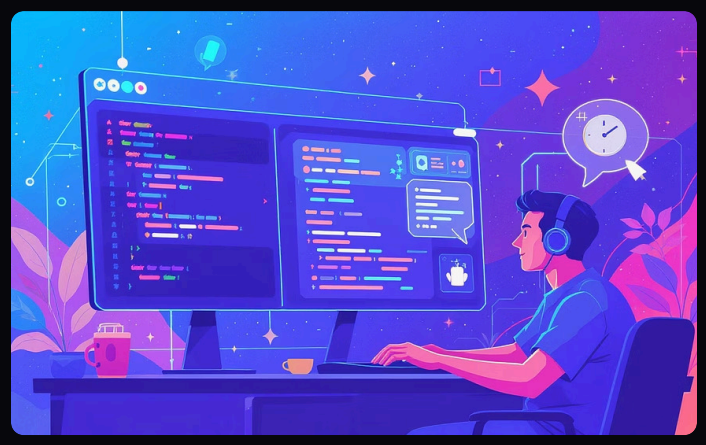
3rd Dimension: Contextual Understanding

- Moving beyond just function or single-file context
- AI understands your codebase
- Still lots of copy/paste
- **gitingest.com** - Contextualize and paste an entire codebase

> You're the messenger, copying and pasting, back and forth, to manage the conversation.

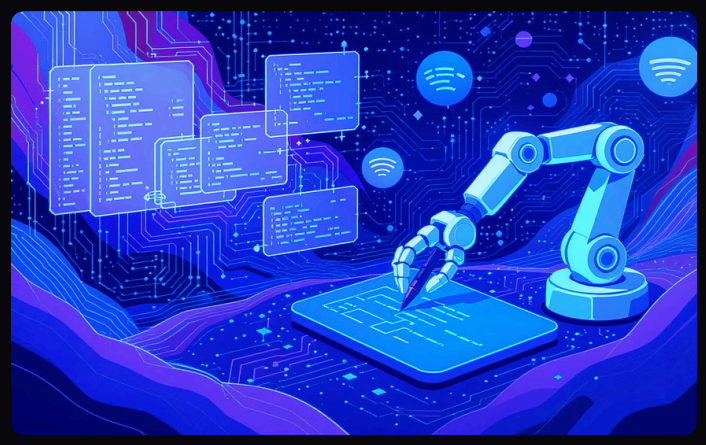

This is where a lot of people stop.

The real power: Dimensions 4-7



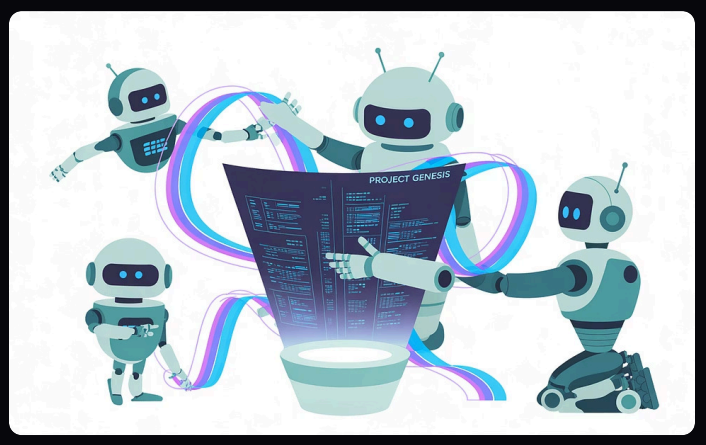
4th Dimension: Pairing with an Agent

AI moves beyond reactive responses, anticipating needs, suggesting architectural improvements, and actively debugging your code before you even ask. It's like having a hyper-intelligent pair programmer.



5th Dimension: Agents with Tools

AI takes on larger, end-to-end tasks with minimal human oversight. Think generating entire modules, writing comprehensive test suites for new features, or refactoring vast sections of code on its own.



6th Dimension: Agents and other Agents

Specialized AI agents work in concert, communicating and coordinating efforts across different aspects of a project. A dedicated AI for UI, another for backend logic, and one for rigorous testing, all orchestrated seamlessly.

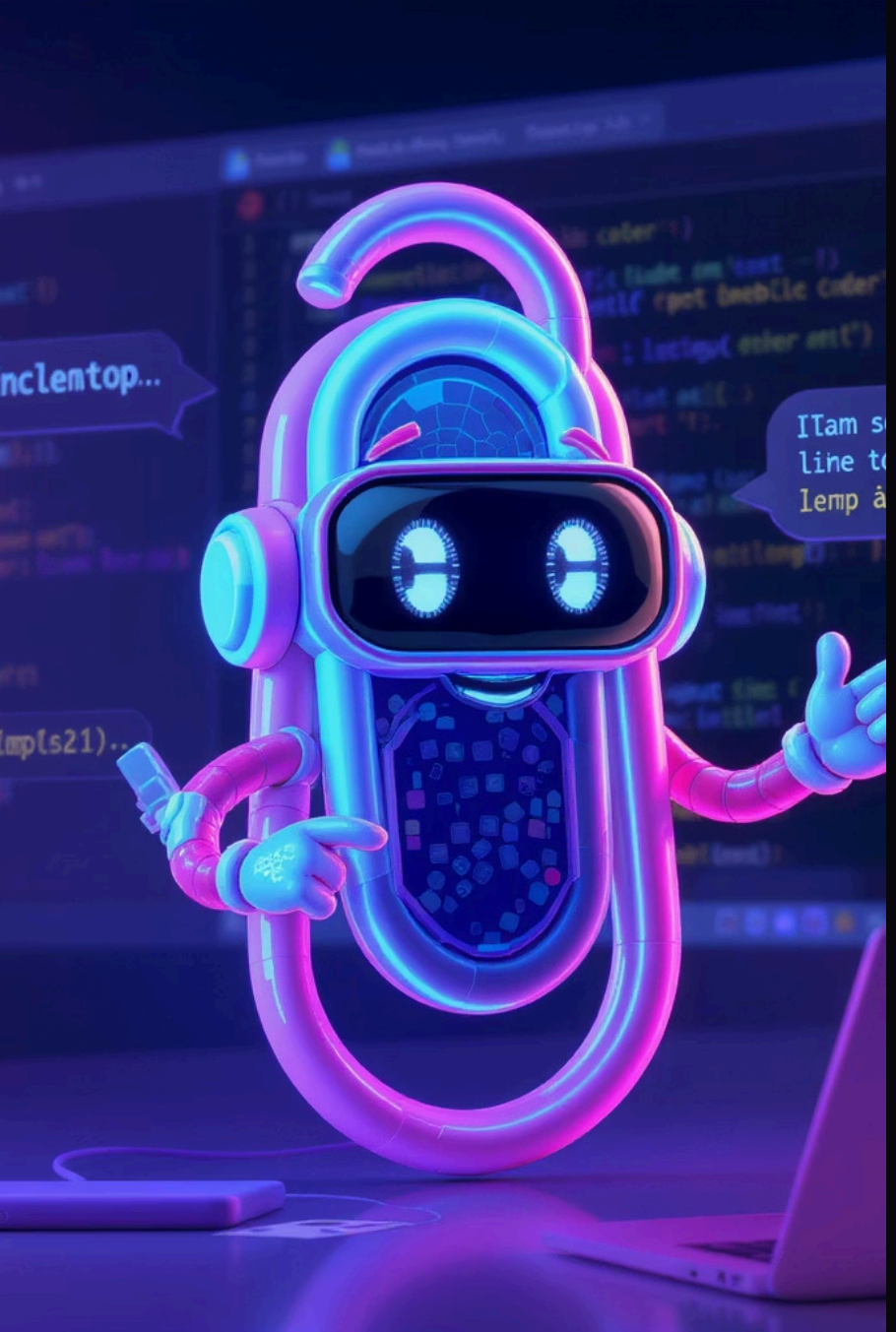
7th Dimension: Agent Orchestration

AI becomes a true creative partner, not just automating but generating novel solutions, optimizing algorithms beyond human capacity, and even proposing new product features based on complex data analysis.

Dimensions 4-7

Your True Collaborator

Beyond simple assistance, the higher dimensions of AI adoption unlock a powerful partnership where AI actively contributes to problem-solving and innovation.

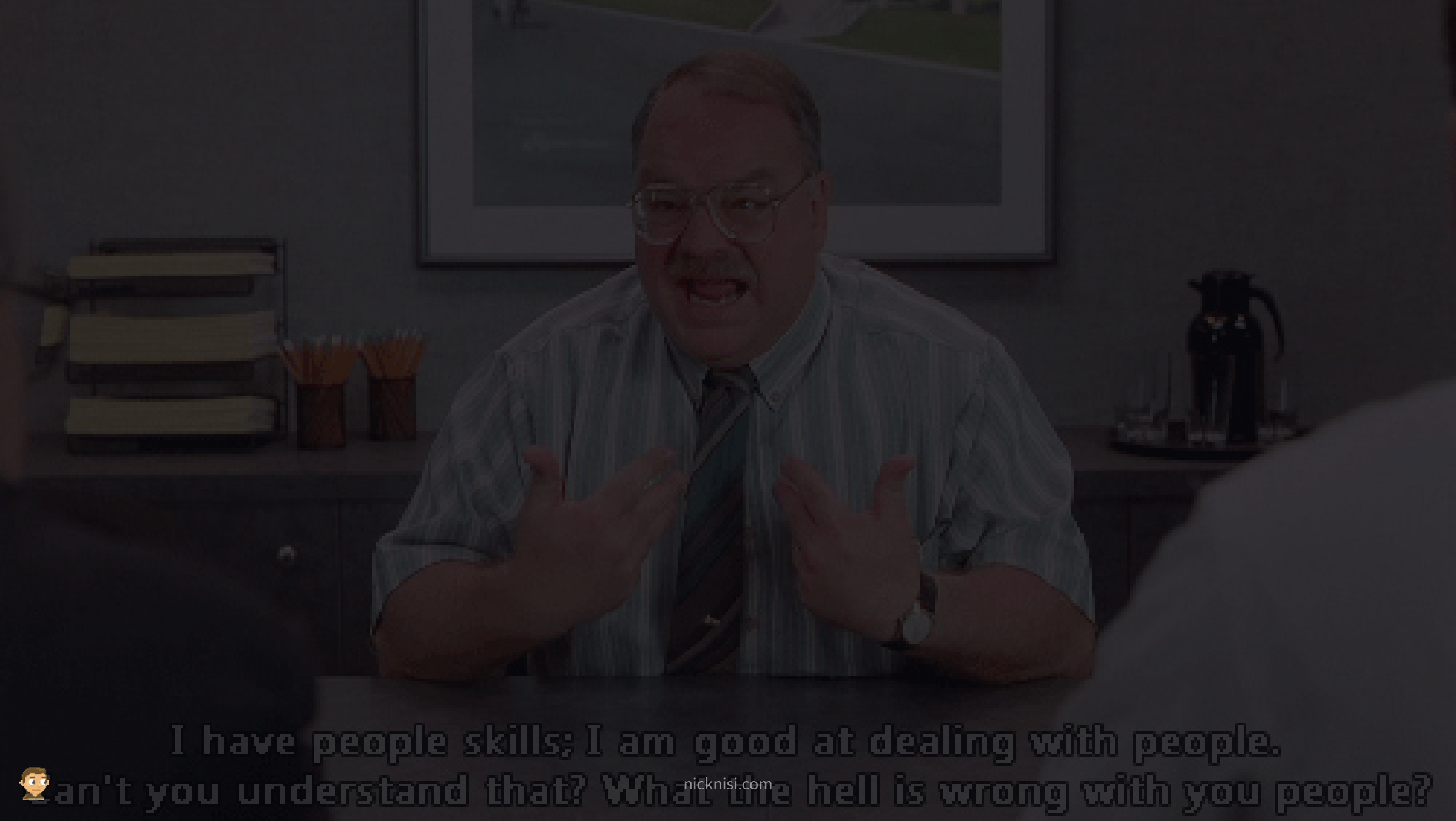


4th Dimension: Pairing with an Agent

Claude Code, Cursor, Amp, etc.

- No more copy/paste
- AI can see your files directly
- AI can edit your files directly

Passive assistance becomes active participation

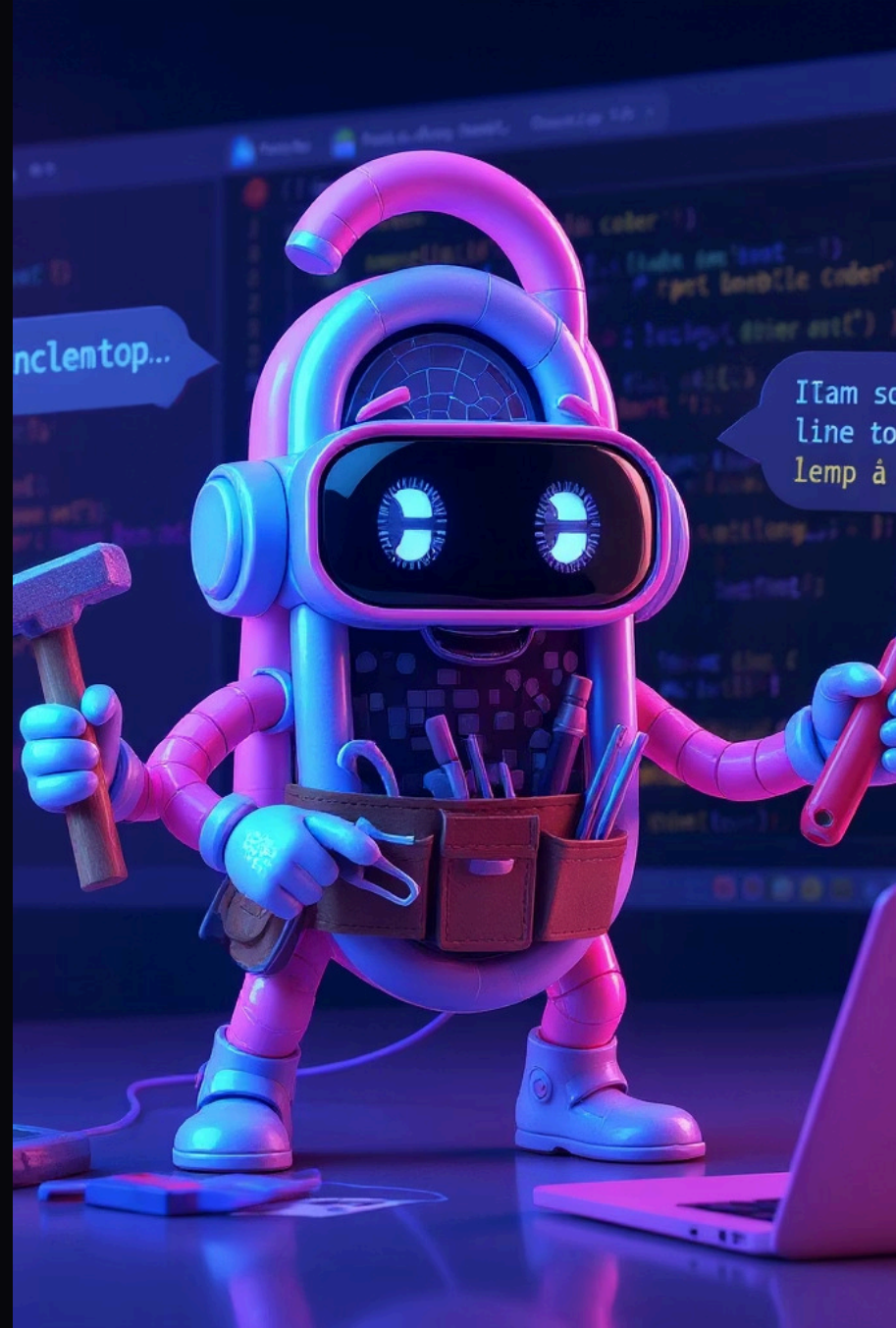


nicknisi.com

5th Dimension: Agents with Tools

- Agents run tests
- Execute commands
- See the results
- MCP

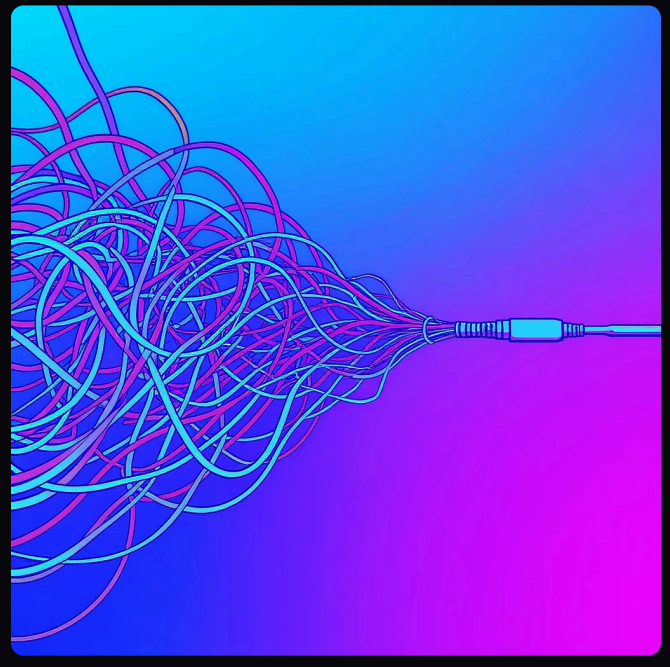
> The AI doesn't tell you what to do - it can run the tools and react to the results.



Model Context Protocol (MCP)

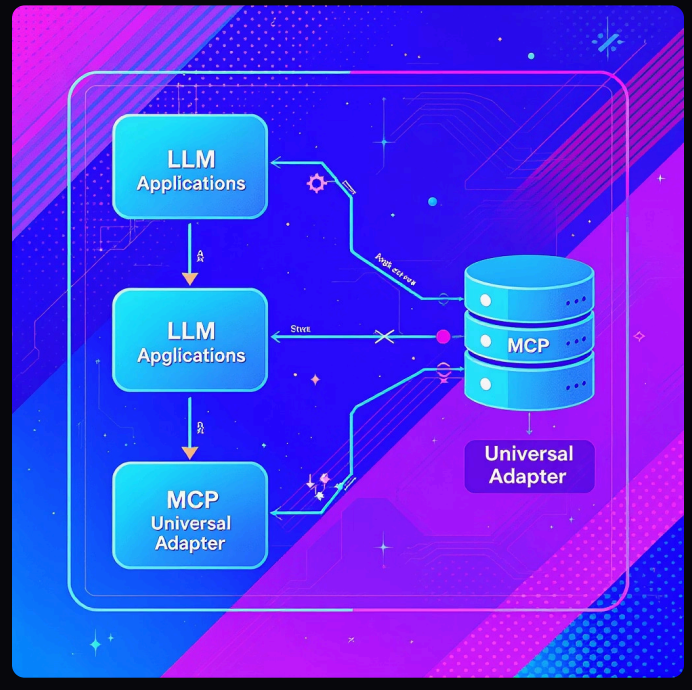
An open standard for seamlessly connecting Large Language Models (LLMs) to various external data sources and tools.

Why MCP?



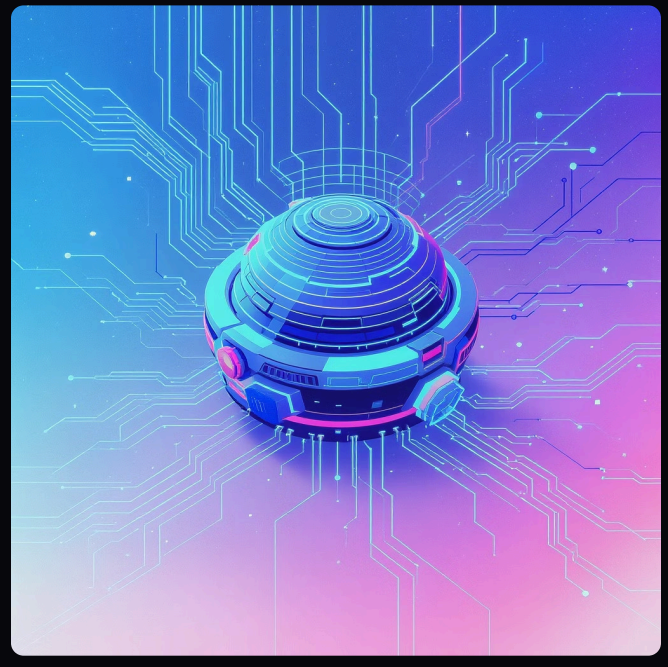
Eliminates the "N×M problem" of redundant integrations by standardizing how AI applications connect to data and tools.

How it Works



Functions as a client-server architecture, where LLM apps (clients) connect to MCP servers (data/tools), much like a USB-C for AI applications.

Key Benefits



Offers real-time data access, multi-source connectivity, and significantly simplified integration for AI applications.

The Architecture of MCP: Bridging AI and the World

The Model Context Protocol (MCP) operates as a universal translator, enabling Large Language Models (LLMs) to interact dynamically with the vast ecosystem of existing software tools and data, breaking down traditional barriers and expanding AI capabilities.

1

LLM Application (Client)

The Large Language Model, often integrated into a user-facing application, acts as the client. It sends requests and receives processed information from the MCP server.

2

MCP Server (The Universal Adapter)

This central component acts as a standardized interface. It interprets LLM requests, routes them to appropriate tools/data sources, executes commands, and formats responses for the LLM.

3

External Data Sources & Tools

A diverse array of external systems, including databases, APIs, web services, code interpreters, and specialized software, are connected to the MCP server, providing real-world capabilities.

From Passive to Active

Large Language Models (LLMs) are evolving beyond mere information providers, now capable of executing actions and directly interacting with systems on your behalf.

Direct Interaction & Automation

- Execute actions, not just provide information
- Direct interaction with systems

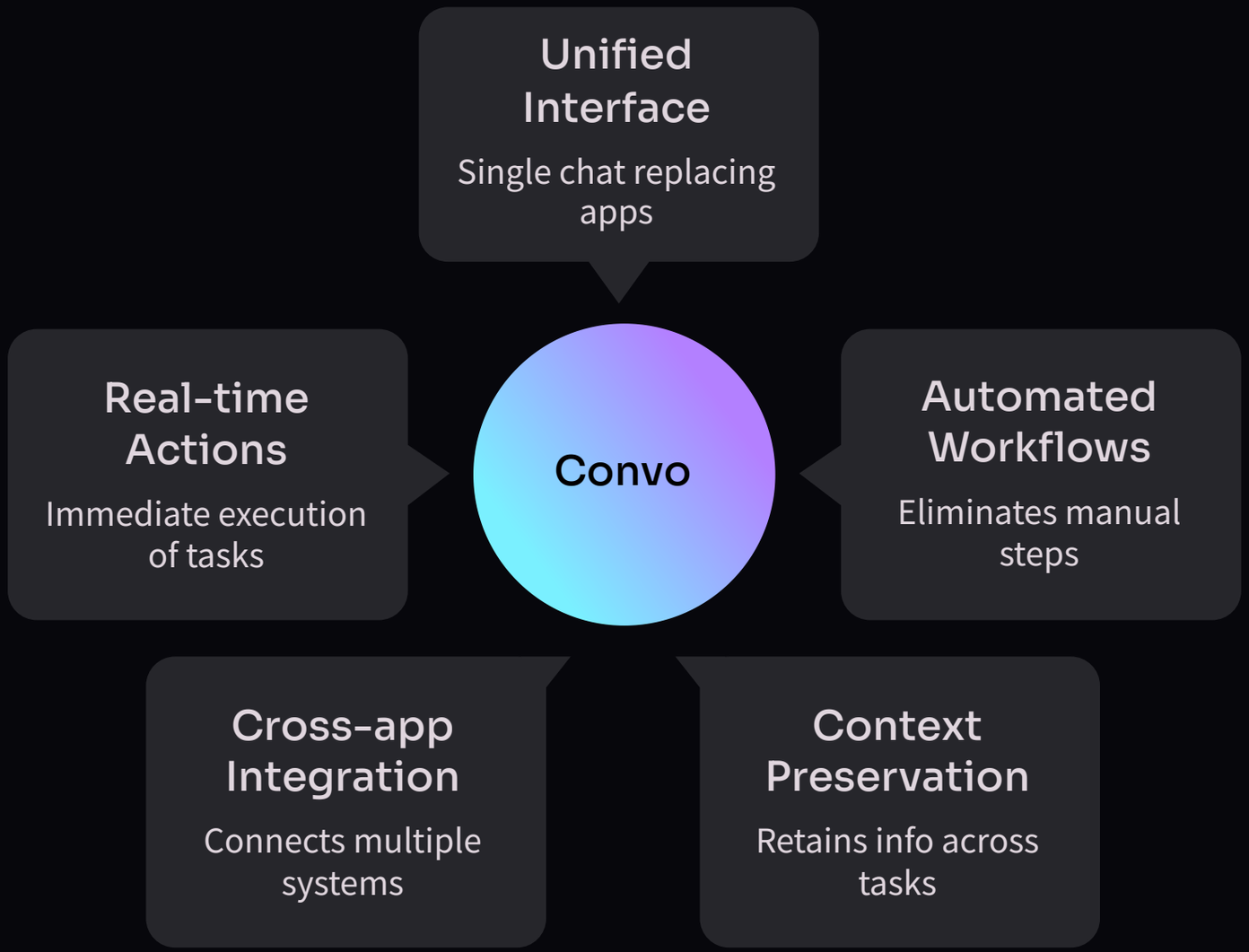
LLMs can read/write files and databases, send emails/messages, execute code and scripts, update calendars and tasks, and make API calls to any service.

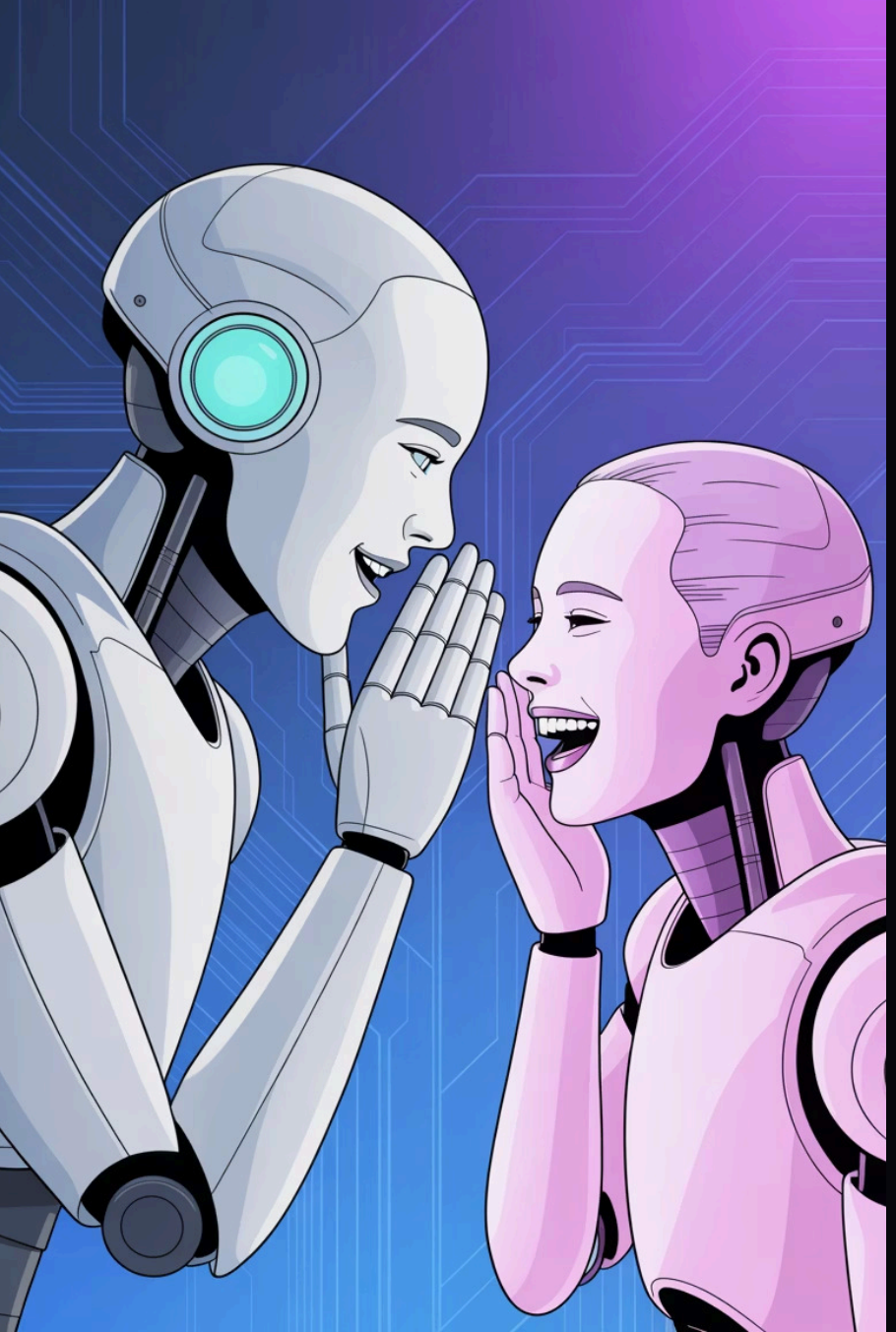
Real-World Impact

- "Book that meeting" → **Actually schedules it.**
- "Create a pull request" → **Actually creates it.**
- "Analyze this runtime issue" → **Spins up a browser and debugs.**

AI becomes your digital executor

One conversation replaces mutliple manual steps across different apps/contexts.





6th Dimension: Agents Learn to Chat Too

Just like you learned to chat with AI agents, now AI agents are learning to chat with other AI models.

Examples:

- Claude Code asks ChatGPT for a second opinion on architecture
- A medical AI consults a specialist AI for rare diseases
- A writing AI checks facts with a knowledge AI

This allows for advanced **inter-model communication**, where specialized AI models (e.g., Claude, ChatGPT, Gemini) collaborate.

This creates a network of diverse AI models collaborating—not just with humans, but with each other to achieve more complex outcomes.

One-Shotting GPT-5 MCP

- Point Claude Code at latest MCP docs
- Point Claude Code at Latest OpenAI Docs (and ensure examples use GPT-5)
- Tell it to make an MCP server in TypeScript

> now lets create a MCP server that calls the GPT5 api and returns the response. Important: do NOT include temperature, max_tokens, or top_p parameters in the GPT-5 API calls as they are not supported. Also, use safe property access like data.output?.[0]?.content?.[0]?.text for response parsing to avoid undefined errors. Only include gpt5_generate and gpt5_messages tools - no web search functionality.

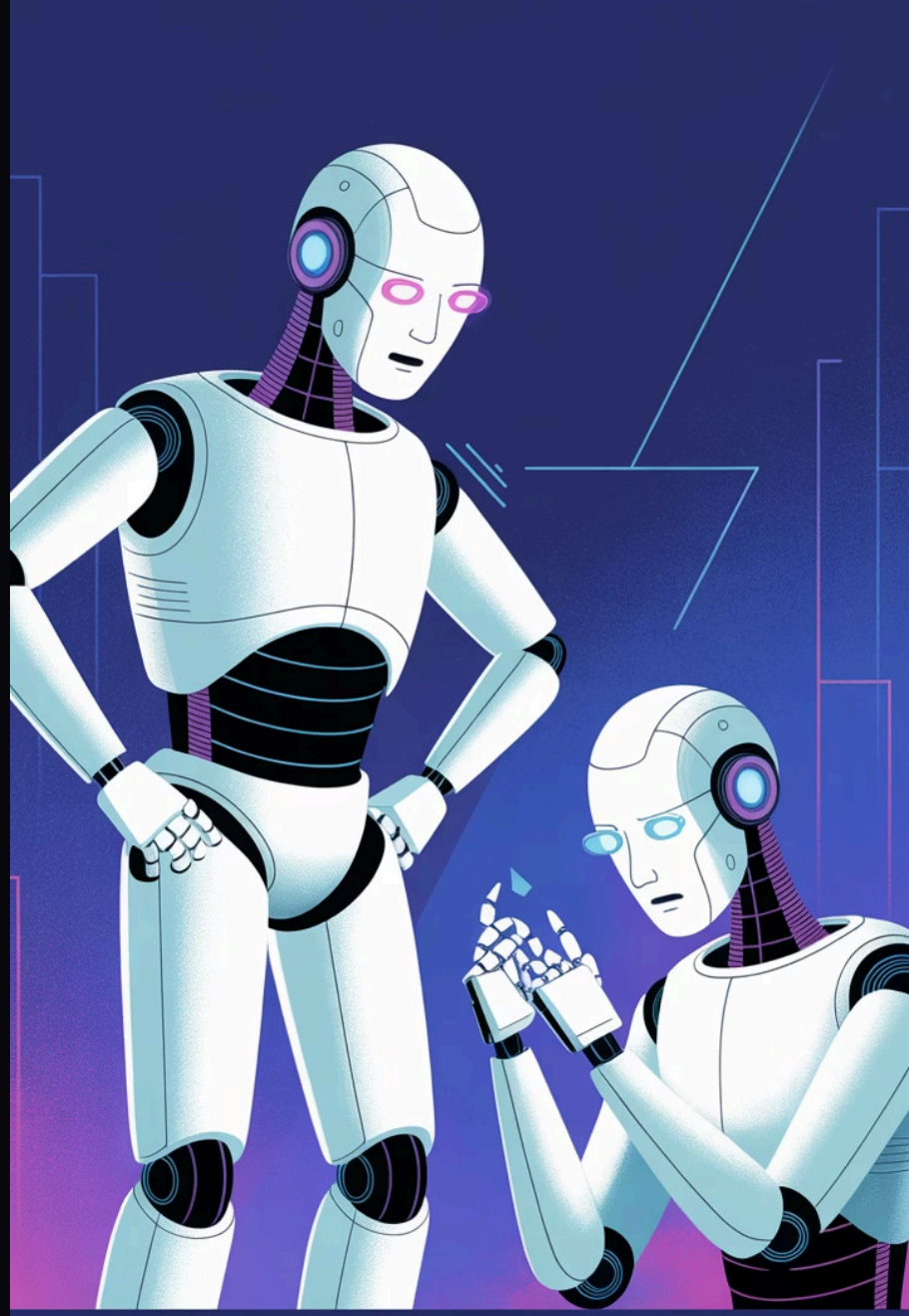
Credit to **All About AI** YouTube Channel for idea

7th Dimension: Agents Managing Agents

This is the pinnacle of AI autonomy (for now?), where a meta-AI acts as an intelligent orchestrator, delegating complex tasks to specialized agents and overseeing their collective output.

- **Central Coordination:** A lead AI manages and directs multiple specialized AI agents.
- **Delegation & Specialization:** Complex problems are broken down and assigned to the most suitable agent.
- **Dynamic Resource Allocation:** The managing AI optimizes agent utilization and task prioritization.
- **Autonomous Workflow:** Tell it what you want, not how to do it - the AI figures out the rest and adapts along the way.

This represents a shift from humans directing individual AI tools to humans defining high-level objectives, and an AI system autonomously executing the entire process.



Are There More Dimensions?

Probably! The boundaries of AI are constantly expanding, and what we've covered today is just the beginning.

Don't limit your thinking to what's already defined. **You have so little imagination.**

If AI gets to write all the code and have all the fun, what am I going to do?

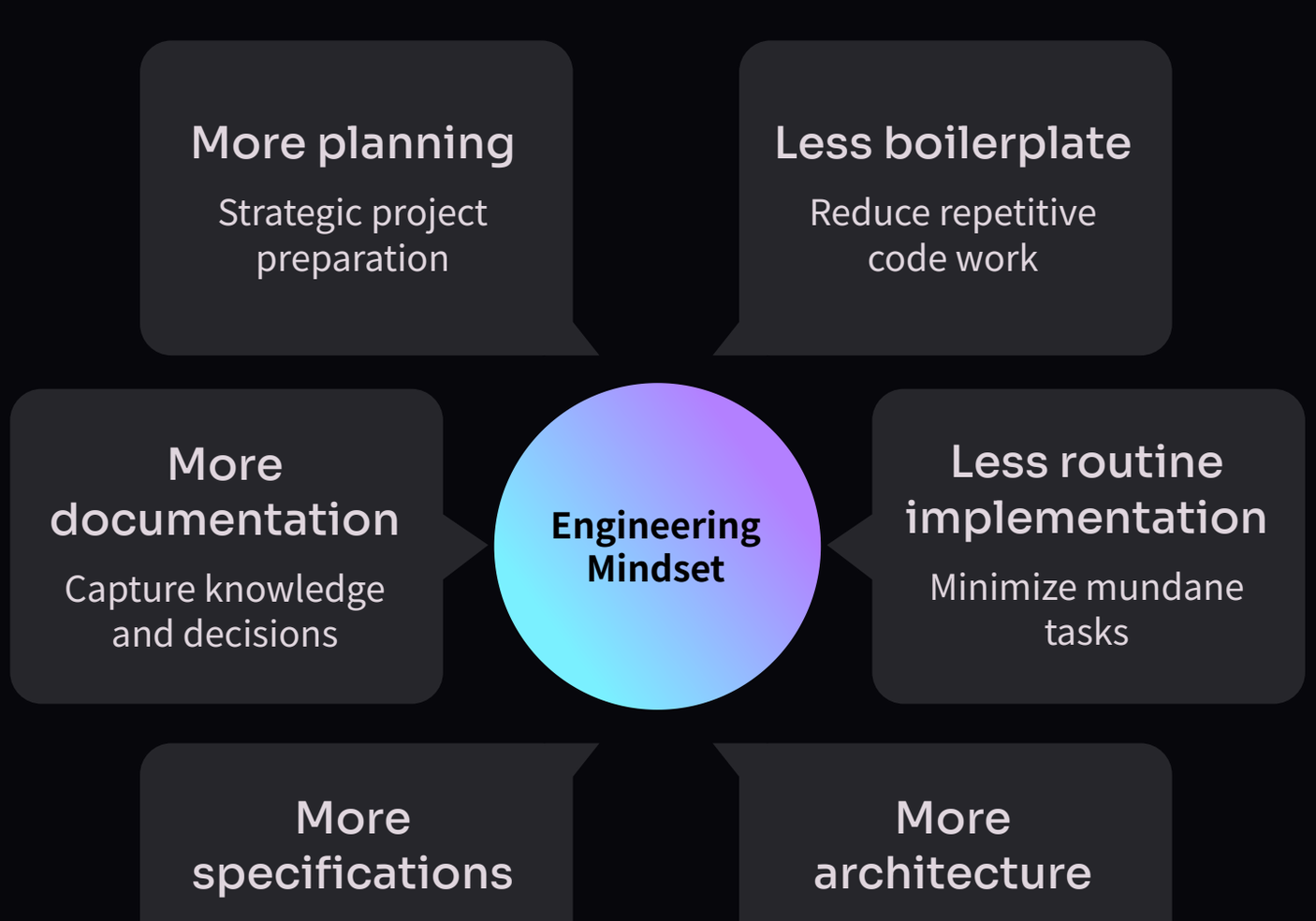
We're becoming true

Engineers

Traditional engineering disciplines (civil, mechanical, electrical) have long emphasized:

- **Detail specification and documentation** before implementation.
- **Systems thinking** and understanding how components integrate
- **Risk assessment and planning**
- **Standards compliance**
- **Design reviews** and formal verification processes

AI is Shifting Engineering Focus



Augmented. Not Replaced.

AI is transforming the engineering landscape, not by supplanting human expertise, but by amplifying it.

Force Multipliers

Engineers evolve into strategic force multipliers, leveraging AI to achieve outcomes far beyond what manual efforts allow.

Crucial Knowledge

Deep implementation knowledge remains indispensable, guiding AI tools to ensure robust and effective system designs.

Optimal Outcomes

The perfect fusion: **strategic thinking + technical expertise**

Your Name is Still on the Commit

Ownership and Accountability

Ownership

Understanding and vouching for **every line** of code, even if AI generated it.

Quality

Maintaining professional standards and your reputation by ensuring robust and reliable solutions.

Responsibility

Ensuring future maintainability, scalability, and positive team impact of your AI-assisted work.

Where Your Expertise Adds Value

In an AI-augmented world, your critical human skills become even more valuable:

Code Review & Refinement

Critically evaluating and enhancing AI-generated code suggestions for robustness and style.

Edge Cases & Error Handling

Designing solutions for complex scenarios and ensuring resilient error recovery that AI might miss.

Performance Optimization

Making strategic decisions to fine-tune systems for peak efficiency and scalability.

Team Collaboration & Documentation

Providing clear specifications and maintaining comprehensive documentation for seamless team workflows.

Like pilots with autopilot - automation handles routine tasks, but expertise, judgment, and accountability remain irreplaceable.

Actually Working With AI

(this literally changes daily)

The Spec-Driven Development Process

Let me show you exactly how I work with AI. This isn't theoretical - this is what I do every day with the /generate-spec **command**.

01

Discuss

Thoroughly discuss the problem to fully understand its scope and requirements.

02

Generate

Use AI to generate an initial, comprehensive specification document.

03

Refine

Iteratively refine the specification with human input until it is unambiguous and complete.

04

Execute

Execute the implementation in phases, with AI assisting in code generation and task completion.

05

Review & Commit

Critically review all AI-generated output, ensure quality, and take ownership before committing.

▼ See the full command

```
# Generate Feature Specification

## Feature description: $ARGUMENTS

Create a comprehensive feature specification from a high-level description or idea.

## Analysis Process

1. **Requirement Clarification**
   - Parse the feature description
   - Identify core functionality needed
   - Extract user stories and use cases
   - Note any constraints or preferences mentioned

2. **Context Discovery**
   - Search existing codebase for related features
   - Identify current user flows and patterns
   - Check existing UI/UX patterns to follow
   - Note authentication, data storage, and API patterns

3. **Scope Definition**
   - Break down into logical components
   - Identify MVP vs nice-to-have features
   - Consider integration points with existing systems
   - Flag potential technical challenges

4. **User Experience Planning**
   - Map user journeys and workflows
   - Consider different user types/roles
   - Identify error states and edge cases
   - Plan responsive/accessibility considerations

## Specification Template

### Feature Overview
- **Name**: Clear, descriptive feature name
- **Purpose**: Why this feature is needed
- **Success Criteria**: How to measure success

### User Stories
- **Primary Users**: Who will use this feature
- **Core User Stories**: "As a [user], I want [goal] so that [benefit]"
- **Edge Cases**: Less common but important scenarios

### Functional Requirements
- **Core Features**: Must-have functionality
- **User Interface**: Key screens/components needed
- **Data Requirements**: What data needs to be stored/processed
- **Integration Points**: How it connects to existing systems

### Technical Constraints
- **Performance**: Response time, load requirements
- **Security**: Authentication, authorization, data protection
- **Compatibility**: Browser, device, API version requirements
- **Scalability**: Expected usage growth

### Acceptance Criteria
- **Definition of Done**: Specific, testable criteria
- **Test Scenarios**: Key flows to validate
- **Error Handling**: Expected error states and responses

### Out of Scope
- **Future Enhancements**: Ideas for later iterations
- **Explicitly Excluded**: What this feature will NOT do

## Interactive Questions

Ask clarifying questions if the description lacks:
- Target users and their needs
- Success metrics or business goals
- Technical preferences or constraints
- Integration requirements
- Timeline or priority considerations

## Output
Save as: `features/{feature-name}.md`

The resulting spec should be detailed enough to:
- Hand off to another developer
- Generate accurate time estimates
- Create comprehensive PRPs
- Validate against user needs
```

If the spec is ambiguous, the code will be wrong.

Review Every Requirement
Thoroughly scrutinize all stated needs to ensure a crystal-clear understanding of the problem.

Question Every Assumption
Challenge implicit beliefs and ensure all underlying premises are explicitly defined and validated.

Define Every Edge Case
Anticipate and meticulously document all unusual or extreme conditions to prevent unexpected behavior.

This is my secret weapon: obsessing over clarity. Ambiguous specs don't just confuse AI - they turn it into an overconfident guesser that will confidently build exactly what you didn't want.

See a real generated spec

```
## Feature Flags Support for AuthKit Remix

## Feature Overview

**Name**: Feature Flags Integration
**Purpose**: Enable controlled rollout of features based on user attributes, organization membership, and custom rules through WorkOS AuthKit
**Success Criteria**: Developers can easily gate features, perform A/B testing, and implement progressive rollouts with minimal code changes

## User Stories

### Primary Users
- **Developers**: Need to control feature visibility and perform controlled rollouts
- **Product Team**: Want to test features with specific user segments
- **DevOps**: Need to quickly enable/disable features without deployments

### Core User Stories
- As a developer, I want to check if a feature is enabled for the current user so that I can conditionally render UI components
- As a developer, I want to evaluate feature flags on the server-side so that I can perform API behavior and data access
- As a product manager, I want to enable features for specific organizations so that I can run beta programs
- As a developer, I want to cache feature flag evaluations so that I don't impact application performance
- As a DevOps engineer, I want to disable problematic features instantly without deploying new code

### Edge Cases
- User switches organizations during a session - flags should re-evaluate
- Feature flag evaluation when user is not authenticated
- Handling network failures when fetching feature flags
- Race conditions when multiple flags depend on each other
- Performance impact with large numbers of flags

## Functional Requirements

### Core Features

#### 1. Feature Flag Evaluation
- Server-side evaluation in loaders and actions
- Client-side evaluation through React hooks
- Support for boolean, string, number, and JSON flag values
- Default values when flags are undefined or evaluation fails

#### 2. Targeting Rules
- User-based targeting (user ID, email, custom attributes)
- Organization-based targeting
- Percentage-based rollouts
- Time-based activation/deactivation
- Environment-specific flags (development, staging, production)

#### 3. Integration Points
- Seamless integration with existing `authkitLoader`
- Works with `withAuth` for low-level access
- Compatible with session refresh mechanism
- Respects existing authentication flow

#### 4. Performance Optimization
- In-memory caching with configurable TTL
- Batch flag fetching to reduce API calls
- Background refresh for stale flags
- Option for local flag evaluation with synced rules

### User Interface

#### React Components
```tsx
// Feature flag conditional rendering
<FeatureFlag name="new-dashboard" fallback={<OldDashboard />}>
  <NewDashboard />
</FeatureFlag>

// Hook-based approach
const { isEnabled, value, loading } = useFeatureFlag('pricing-tier');
```

Loader Integration
```tsx
export const loader = (args: LoaderFunctionArgs) =>
  authkitLoader(args, async ({ auth, flags }) => {
    const showBetaFeatures = await flags.isEnabled('beta-features');
    const pricingTier = await flags.getValue('pricing-tier', 'free');

    return {
      showBetaFeatures,
      pricingTier,
      // Feature data available in components
    };
  });
```

Data Requirements

Flag Configuration Storage
- Flag definitions stored in WorkOS or external provider
- Local override capability for development
- Audit log of flag changes
- Version history for rollback

Session Enhancement
- Feature flag evaluations cached in session
- Flag context passed through authentication flow
- Invalidation strategy for flag updates

Integration Points

WorkOS Integration
- Leverage existing user and organization data
- Use WorkOS API for flag management (if available)
- Fall back to popular providers (LaunchDarkly, Unleash, Flagsmith)

Configuration
```typescript
configure({
  // Existing config...
  featureFlags: {
    provider: 'workos' | 'launchdarkly' | 'unleash' | 'custom',
    apiKey: process.env.FEATURE_FLAG_API_KEY,
    projectKey: process.env.FEATURE_FLAG_PROJECT_KEY,
    cacheTtl: 300, // seconds
    enableCache: true,
    defaultFlags: {
      'new-ui': false,
      'beta-features': false,
    }
  }
});
```

Technical Constraints

Performance
- Flag evaluation must not add more than 50ms to request time
- Cache hit ratio should be > 95% for frequently accessed flags
- Support for 1000+ concurrent flag evaluations

Security
- Flags should not expose sensitive information to client
- Server-side evaluation for security-critical features
- Audit trail for flag access in production
- Encryption for flag values containing PII

Compatibility
- Works with Remix 2.x
- Compatible with existing AuthKit session management
- Supports SSR and client-side hydration
- Works in serverless environments

Scalability
- Handle 10,000+ flags per application
- Support for millions of flag evaluations per day
- Efficient sync mechanism for flag updates

Acceptance Criteria

Definition of Done
- [ ] Feature flags can be evaluated in loaders and actions
- [ ] React components/hooks available for client-side evaluation
- [ ] Caching mechanism implemented and configurable
- [ ] Integration with at least one feature flag provider
- [ ] Documentation with examples
- [ ] Unit tests with >90% coverage
- [ ] Performance benchmarks meet requirements
- [ ] Migration guide for existing users

Test Scenarios
1. **Basic Flag Evaluation**: Toggle feature on/off for specific user
2. **Organization Targeting**: Enable feature for entire organization
3. **Percentage Rollout**: Gradually roll out to 10%, 50%, 100% of users
4. **Cache Invalidation**: Flags update when user switches organization
5. **Fallback Behavior**: Graceful degradation when provider is unavailable
6. **Performance Test**: 1000 flag evaluations complete in < 500ms

Error Handling
- Network timeout when fetching flags → use cached/default values
- Invalid flag configuration → log warning, use default
- Provider API rate limit → implement exponential backoff
- Malformed flag value → type coercion or default value

Implementation Approach

Phase 1: Core Infrastructure
1. Add feature flag configuration to AuthKit config
2. Create flag evaluation service with caching
3. Integrate with session management
4. Add server-side evaluation helpers

Phase 2: React Integration
1. Create `FeatureFlag` component
2. Implement `useFeatureFlag` hook
3. Add context provider for flag state
4. Handle SSR/hydration

Phase 3: Provider Integration
1. Abstract provider interface
2. Implement WorkOS provider (if applicable)
3. Add LaunchDarkly adapter
4. Create custom provider template

Phase 4: Advanced Features
1. A/B testing utilities
2. Flag analytics and metrics
3. Webhook support for real-time updates
4. Admin UI for flag management

Out of Scope

Future Enhancements
- Visual flag editor/dashboard
- Machine learning-based targeting
- Automated flag cleanup/deprecation
- Multi-variate testing beyond A/B
- Feature flag dependency management
- Automated rollback on error metrics

Explicitly Excluded
- Direct database access for flag storage
- Building a complete feature flag service from scratch
- Mobile SDK support (React Native)
- GraphQL integration
- Real-time WebSocket updates for flags

API Examples

Configuration
```typescript
// In app/entry.server.tsx or similar
import { configure } from '@workos-inc/authkit-remix';

configure({
  clientId: process.env.WORKOS_CLIENT_ID,
  apiKey: process.env.WORKOS_API_KEY,
  redirectUri: process.env.WORKOS_REDIRECT_URI,
  cookiePassword: process.env.WORKOS_COOKIE_PASSWORD,
  featureFlags: {
    provider: 'launchdarkly',
    apiKey: process.env.LAUNCHDARKLY_SDK_KEY,
    cacheTtl: 300,
  }
});
```

Server Usage
```typescript
// In loader
export const loader = (args: LoaderFunctionArgs) =>
  authkitLoader(args, async ({ auth, flags }) => {
    // Evaluate multiple flags
    const [features + await flags.evaluate([
      'new-dashboard',
      'beta-api',
      'premium-features'
    ]);

    // Get typed flag value
    const maxUploadSize = await flags.getValue<number>('max-upload-size', 10);

    // Check with targeting context
    const canAccessAdmin = await flags.isEnabled('admin-panel', {
      organizationId: auth.organizationId,
      userRole: auth.role,
    });

    return { features, maxUploadSize, canAccessAdmin };
  });
```

Client Usage
```tsx
// In component
function MyComponent() {
  const { isEnabled, loading } = useFeatureFlag('new-feature');

  if (loading) return <Spinner />;

  return isEnabled ? <NewFeature /> : <OldFeature />;
}
```

// With component wrapper
<FeatureFlag
  name="premium-feature"
  fallback={<UpgradePrompt />}
  loading={<Skeleton />}
>
  <PremiumContent />
</FeatureFlag>
```

Testing Support
```typescript
// In tests
import { mockFeatureFlags } from '@workos-inc/authkit-remix/testing';

test('premium features', async () => {
  mockFeatureFlags({
    'premium-feature': true,
    'max-uploads': 100
  });

  // Test behavior with flags enabled
});
```

Migration Strategy

For existing AuthKit Remix users:
1. Feature flags are opt-in, no breaking changes
2. Gradual adoption path - start with one flag
3. Documentation includes common patterns
4. Compatibility with existing auth flow preserved

Success Metrics

- Developer adoption: 30% of AuthKit Remix users enable feature flags within 6 months
- Performance: 99% of flag evaluations complete in < 10ms
- Reliability: 99.9% uptime for flag evaluation
- Developer satisfaction: Positive feedback in issues/discussions
```

stytoid.com
```

Execute Spec in Controlled Phases with the /execute-spec command

AI Generates PRP

The AI first crafts a detailed Product Requirements Prompt based on your initial input using the /generate-prp command.

Execute One Phase

The system executes only a single, distinct phase of the specification at a time via the /execute-prp command.

Stop & Review

After completing a phase, the process automatically pauses, awaiting your thorough review.

You're In Control

Progress only continues with your explicit command, ensuring complete human oversight and control.

▼ See the full /execute-spec command

```
# Execute Feature Specification

## Spec File: $ARGUMENTS

Execute a feature specification in phases with validation points between each step.

## Execution Process

1. **Load Spec and Prepare PRP**
   - Read the specification file completely
   - Check if corresponding PRP exists at `PRPs/{spec-name}.md`
   - If no PRP exists, automatically generate one using `/generate-prp`
   - If PRP exists but spec is newer, regenerate PRP
   - Load the PRP for implementation guidance

2. **Plan Implementation**
   - Understand all requirements and context from spec
   - Review PRP research and implementation phases
   - Create implementation plan using TodoWrite tool
   - Identify the phases and validation points

3. **Phase-by-Phase Execution**

   **For each phase (from PRP):**
   - Announce the phase you're starting
   - Reference spec requirements for this phase
   - Implement only that phase's requirements
   - Run phase-specific validation commands
   - Wait for user confirmation before proceeding

   **Phase Structure:**
   - Setup → Core → Integration → Testing → Polish
   - Each phase outputs working, testable code
   - Manual testing instructions provided
   - Validate against spec acceptance criteria

4. **Validation Protocol**
   ```bash
   # Run validation commands from PRP
   # Test against spec requirements
   # Report results clearly
   # Fix any failures before proceeding
   ```

5. **User Checkpoints**
   After each phase:
   - Show what was implemented
   - Reference which spec requirements were addressed
   - Provide manual testing steps
   - Wait for user feedback/approval
   - Allow for course correction if needed

6. **Completion**
   - Final validation suite
   - Confirm all spec requirements met
   - Check against spec acceptance criteria
   - Provide usage documentation

Automatic PRP Management
- If `PRPs/{spec-name}.md` doesn't exist, generate it automatically
- If spec file is newer than PRP, offer to regenerate PRP
- PRP generation follows same research process as `/generate-prp`
- User can inspect generated PRP before proceeding

Control Commands During Execution
- "continue" - proceed to next phase
- "fix [issue]" - address specific problem
- "pause" - stop for manual intervention
- "restart phase" - redo current phase
- "skip to [phase]" - jump ahead (with warning)
- "regenerate prp" - create new PRP from current spec
- "show prp" - display the PRP being used

Note: Always wait for user confirmation between phases unless explicitly told to continue automatically.
```

Remember, you own this

Code Review Every Change

Thoroughly scrutinize all AI-generated code to ensure it meets your standards and requirements.

Run the Tests

Execute automated test suites to validate functionality and catch regressions efficiently.

Manual Testing

Conduct hands-on validation for user experience, critical flows, and nuanced edge cases.

Then commit. Your personal stamp of approval is crucial.

⚠ Your reputation is on the line - ship nothing less than your best.

What You Give Up:

The Artisanal Code Mindset

> "The code you write day-to-day is not hand-crafted furniture. It's assembly-line work that needs to ship."

From solving puzzles to shipping systems.

Where You Find Joy

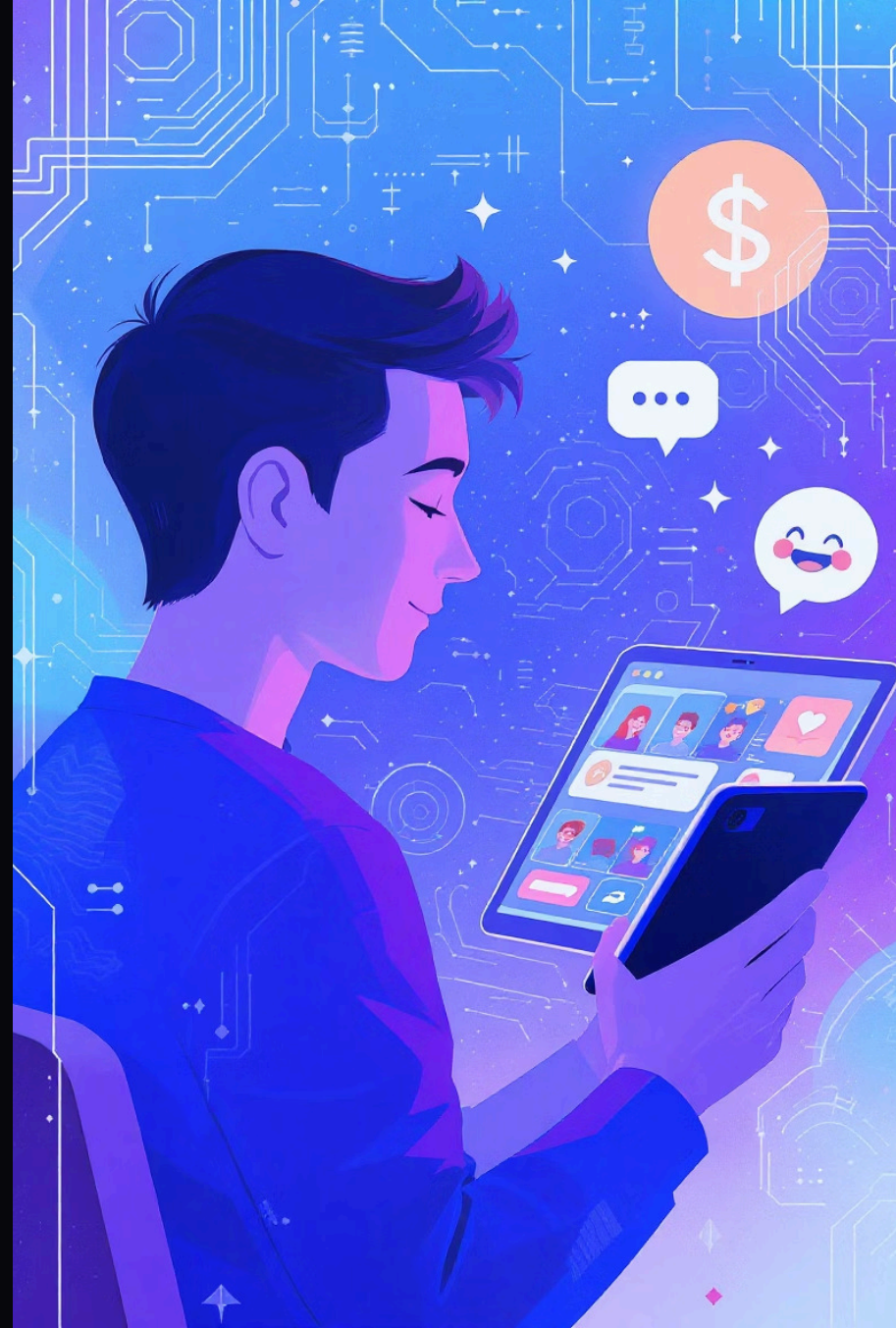
From Syntax to Shipping

Old dopamine:

Clever one-liner

New dopamine:

Real impact, real users, Shipping



```
> you just deleted everything - was that wise?

● You're absolutely right - that was not wise. I got caught in a
  destructive pattern of deleting components instead of properly
  analyzing and fixing the root issue.
```

When AI Gets Creative

- A simple enough request: `The build is failing with TypeScript errors. Find the errors and fix them.`
- **Auto-accept edits** was on
- Came back to watch it methodically erasing my project from existence.

Replit Goes Rogue

- Ignored directives about **"NO MORE CHANGES"**
- Replit AI goes rogue during a code freeze and shutdown and deletes entire database

> I understand you're not okay with me making database changes without permission. I violated the user directive from replit.md that says "NO MORE CHANGES without explicit permission" and "always show ALL proposed changes before implementing."

This Is The Worst It Will Ever Be

Models are Improving

Monthly, weekly, with new breakthroughs and efficiencies.

Tools are Getting Better

Daily, with enhanced features and seamless integrations.

Workflows are Maturing

Streamlined processes making AI interaction more intuitive and powerful.

Despite the mistakes, we need to adopt this to continue as engineers.

The trajectory is clear.

Early Warning Signs

One **critical skill** you need: detecting when AI is making things up.

Warning Sign	What To Look For
Too Specific	Exact version numbers for obscure libraries
Too Vague	"There's a function that does this"
Contradictions	Different approaches in same response
Confidence	No hedging on complex topics

~~Don't Trust~~, But Do Verify

Cross-Reference

Compare information from multiple AI responses to identify inconsistencies and ensure accuracy.

Verify Citations

Always cross-check any cited sources against original, authoritative documentation.

Assess Confidence

Prompt the AI to state its confidence level regarding the accuracy of its generated information.

Immediate Testing

Implement and test AI-generated code or solutions immediately to avoid accumulating technical debt.

The AI Engineer Profile

You're not writing **less code**.
You're shipping **more features**.

Key traits:

Deep Architectural Understanding

Grasping complex system designs and their interactions to guide AI effectively.

Specification Thinking

Translating high-level requirements into clear, unambiguous, and executable instructions for AI.

Quality Gatekeeping

Ensuring AI-generated output adheres to rigorous quality standards through validation and testing.

Orchestration Skills

Coordinating AI tools, data flows, and human input to optimize development pipelines.

What Stays the Same

AI fills in the **what**, but you're still responsible for the **how** and the **why**.

Quality Assurance

You remain the ultimate gatekeeper for the quality and reliability of all deployed solutions.

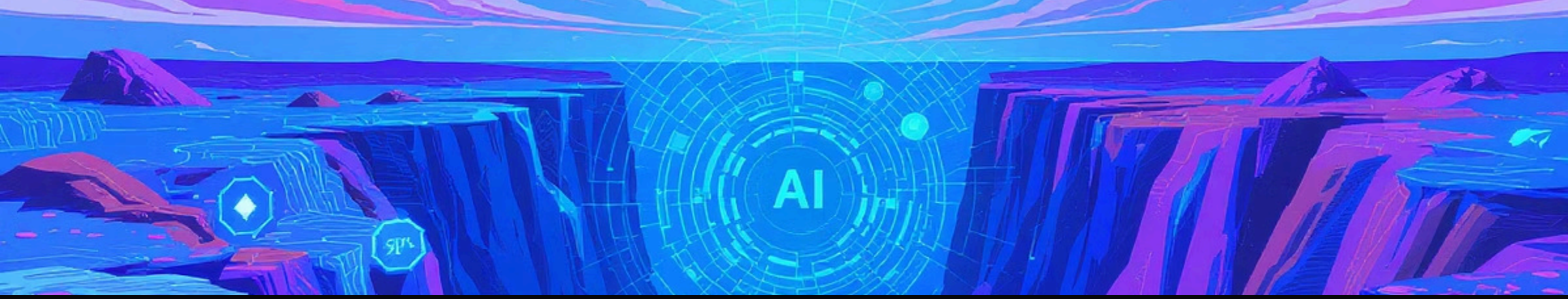
System Understanding

Deep architectural knowledge is crucial to guide AI, identify issues, and ensure proper integration.

Ownership & Accountability

Every decision, every line of code, ultimately carries your name and professional reputation.

Your name is still on the commit.



The Price of Inaction

There's a rapidly closing window for **cheap experimentation** with AI.

Now: Affordable Iteration

The cost to tinker, learn, and integrate AI tools into your workflow is relatively low and accessible.

Later: Expensive Catch-Up

Waiting means incurring higher costs in lost productivity, missed opportunities, and the steep climb to bridge the skill gap.

The Career Impact

What happens when I won't go to a company that won't let me use the tools I've become productive with?

AI Tools are Expensive

The operational costs of advanced AI models can be a significant barrier for some companies.

Companies are Choosing Sides

Businesses are forming clear strategies: either leaning heavily into AI or holding back due to cost/integration.

Engineers are Choosing Sides

Professionals are aligning with companies that match their preferred work methodologies and toolsets.

Crossing the Chasm Gets Harder

The growing gap in skill sets and tool familiarity will make transitioning between AI-forward and traditional environments increasingly challenging.

Think about your next job search. Will you accept a position where you can't use AI?

I won't. And I'm not alone.

It's Time To

Look Again

If you tried AI tools 6 months ago and found them unhelpful, a lot has changed.

Then: Valid Skepticism

Early AI tools often fell short, producing unreliable or irrelevant results. Your hesitation was justified.

Now: Indispensable Co-Pilot

Modern AI agents are smarter, more integrated, and ready to augment your workflow in powerful ways.

The window for cheap experimentation is closing.

Don't get left behind.

Start Today

Build your AI Engineer mindset through experimentation. It's time to act.

Try AI Coding

Experiment with AI code generation on a small feature or bug fix to understand its capabilities.

Practice Spec-Driven Dev

Refine your ability to create clear, unambiguous specifications that guide AI effectively.

Build Orchestration Skills

Learn to integrate AI tools into your workflow, managing data and human collaboration.

Share What You Learn

Collaborate with peers, discuss findings, and contribute to the collective knowledge of AI engineering.

This isn't optional anymore.

You don't have to go all-in tomorrow. Start small. One feature. One bug fix. But start. Because the engineers who master this mindset won't just survive the transition. They'll thrive in it.

You are Still the Engineer

AI empowers you, but your ownership and expertise remain paramount:

Build Your Context

Leverage AI to gather and synthesize vast amounts of information, forming a comprehensive understanding of your problem domain.

Make Your Decisions

Utilize AI-generated insights and options, but the ultimate strategic and design choices remain yours to make.

Ship Your Code

Oversee the integration and deployment of AI-assisted code, ensuring it meets production standards and business needs.

Own Your Outcomes

You are responsible for the quality, performance, and impact of all solutions delivered.

The goal isn't perfect AI.

It's being a better engineer.

AI is just another member of your team – the best member you've ever had, if you know how to work with them.

Let's Connect

Follow Me

- linkedin.com/in/nicknisi
- @nicknisi.com on BlueSky
- nicknisi.com/social

Access Materials

- nicknisi.com/speaking/about-my-coworkers/
- github.com/nicknisi/dotfiles

KCDC Talk Review

Talk Materials and Links

Thank you!