# Cursor Editor

The AI-Powered Code Editor

Revolutionizing Programming with Human-AI Collaboration

## What is Cursor?

- A fork of VS Code with powerful AI features
- Built by team who believed in scaling laws and GPT-4's potential
- Designed for *human-AI collaboration*, not replacement
- Used by hundreds of thousands of programmers

> *"Fast is fun. Programming should be delightful."*

## Core Philosophy

Human Intelligence + Artificial Intelligence = Magic

- **Human:** Memory, analysis, architecture, project management
- **AI:** Efficient text/code generation, pattern recognition
- **Together:** Eliminate "zero entropy" keystrokes

## 🔥 Cursor Tab

The all-knowing autocomplete that predicts your next move

- Predicts entire code changes, not just characters
- Jumps to next logical editing location
- Uses fast MOE models with speculative decoding
- Goal: Press Tab → Make progress

"*How many Tabs can we make someone press?*"

# How Next Move Prediction Works

[Fill-in-the-middle](https://github.com/QwenLM/Qwen2.5-Coder): A generalization of code completion

## What you see:

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    |  # ← cursor here
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(
```

## What the AI sees:

```
<|fim_prefix|>def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    <|fim_suffix|>
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right
<|fim_middle|>
```

Model generates: `left = [x for x in arr if x < pivot]`

## Smart Context System

### What you can add:

- Files & folders
- Code snippets
- Documentation
- Git history
- Terminal output
- Lint errors
- Web pages
- Images

### How it works:

- Semantic codebase indexing
- Merkle tree sync for large repos
- Privacy-preserving embeddings

## 🔍 Codebase Indexing Deep Dive

Semantic understanding of your entire codebase without storing your code

### How it works:

- **File scanning:** Computes Merkle tree of hashes, respects .gitignore/.cursorignore
- **Incremental sync:** Checks every 10 minutes, uploads only changed files
- **Server processing:** Chunks and embeds files, stores in Turbopuffer vector DB
- **Privacy-preserving storage:** Only embeddings + obfuscated paths stored
- **Smart retrieval:** Nearest neighbor search → local file read → context to AI

Learn more

## Apply: From Sketch to Code

Frontier models create rough sketches → Custom models generate precise diffs

```
// You write this rough idea:
function processUsers() {
  // TODO: fetch users from API
  // validate data
  // transform to our format
  // save to database
}

// Cursor Apply generates the actual implementation
async function processUsers() {
  const response = await fetch('/api/users');
  const users = await response.json();

  const validUsers = users.filter(user =>
    user.email && user.name && typeof user.id === 'number'
  );
```

## Intelligent Diff Interface

- **Multiple diff modes:** Optimized for autocomplete vs. large changes
- **Smart highlighting:** Show important changes, gray out repetitive ones
- **Multi-file support:** Coordinated changes across your codebase
- **Background processing:** Shadow workspace for testing changes

"*Code review kind of sucks. We can do much better with language models.*"

## 🤖 Agent Mode

AI that can use tools and make coordinated changes

- **Web search:** Find documentation and examples
- **Terminal execution:** Run commands and tests
- **File system access:** Create, modify, and organize files
- **MCP tools:** Extensible tool ecosystem
- **Planning mode:** Break down complex tasks

## Under the Hood

### Model Ensemble:

- **Claude Sonnet:** Best overall coding performance
- **GPT-4/o1:** Complex reasoning tasks
- **Custom models:** Tab completion, Apply diffs

### Performance Optimizations:

- Speculative decoding for faster generation
- KV cache optimization
- MOE (Mixture of Experts) models
- Cache warming and preemptive requests

## Recommended Workflow

- **Break down tasks:** Decompose features into specific, implementable chunks
- **Add relevant context:** Include related files, docs, and examples
- **Start new chats:** Fresh context for each distinct task
- **Be specific:** Clear prompts get better results
- **Iterate quickly:** Make small changes and build incrementally
- **Review AI output:** You're the architect, AI is the implementer

## Real-World Example

Building mcpbar: A CLI package manager for MCP servers

- **90%+ code generated** by Cursor
- Bootstrapped from open source template
- Used context to understand MCP protocol
- Applied best practices for CLI design
- Iterative development with human guidance

"*Human as project manager, AI as highly productive developer*"

## Current Limitations

- **Bug detection:** Models struggle with finding subtle bugs
- **Large diffs:** Reviewing big changes is still challenging
- **Context limits:** Can't include entire large codebases
- **Domain knowledge:** May lack specific business context
- **Latency:** Some operations still take seconds

But these are improving rapidly! 🚀

## Why Cursor Succeeded

Key insight: Think one step further, pursue excellence

- **Beyond autocomplete:** GitHub Copilot → Cursor predicts next actions + Tab navigation

  *Created the "ambient programming" phenomenon on social media*

- **Beyond chat diffs:** GitHub Copilot → Cursor built in-editor diff interface

  *Faster review and application of AI changes*

- **Custom models:** Self-developed models optimized for core features
- **Fast iteration:** Early adopter of MCP integration and other innovations

## Does Cursor Have a Moat?

### 🏰 Shallow Technical Moat

Code editors have low switching costs - users can easily move to better/cheaper alternatives

### 💎 Real Moat: Execution Excellence

- **Talent:** Few teams can pursue excellence and iterate at this speed
- **Matthew Effect:** Success breeds more success and resources

*"Don't settle for a product that works - build one that users love"*

# The Future of Programming

> *"A human-AI programmer that's an order of magnitude more effective"*

## Human Strengths:

- Intent understanding
- Long-term memory
- Business context
- Architecture decisions

**Weakness:** Slow at code production

## AI Strengths:

- Very productive code writing
- Pattern recognition
- Syntax perfection
- Rapid iteration

**Weakness:** Limited context & memory

- **Abstraction control:** Move up and down the stack seamlessly
- **Natural language:** Sometimes, but not always
- **Background agents:** Autonomous helpers for routine tasks

## 🚀 Embrace the AGI Revolution

AGI will change almost everything

Everything that can be mapped to structural data can be mastered by AI and see massive productivity boosts

- 💻 **Code:** Cursor
- 🎨 **Image Editing:** Adobe, Canva
- 🎬 **Video Generation:** Kling
- 📊 **Presentations:** Gamma

Don't just watch the revolution — be part of it! 🌟

## 🎁 Bonus for Curious Minds

Want to dive deeper into how Cursor works?

- **Peek behind the curtain:** View the leaked prompt from Cursor's agent mode to understand in-depth how it works

  **Bonus:** You can also view the markdown version of the prompt.
- **See AI in action:** This slide deck itself was written in Cursor and co-edited by the Cursor agent! Check out the source code to see how I automate slide creation with AI

*"Meta-programming: Using AI to create slides about AI"* 🤖✨

## Questions?

Let's explore the future of programming together