



Agentic Engineering

Christian Budde [@MeKo-Christian](https://twitter.com/MeKo-Christian)

Einzelner Prompt

ChatGPT, Claude.ai, Gemini, ...

Mensch: Prompt

LLM: Antwort



Mensch liest,
kopiert, prüft,
führt aus, ...

- Eine Frage, eine Antwort
- LLM sieht nur Text – handelt nicht
- Jeder Schritt: Mensch ist Ausführer
- Kontext endet mit der Antwort

→ **Prompt Engineering** = Kunst, den
richtigen Prompt zu formulieren



Agent

[Claude Code, Codex, Gemini CLI, Cursor, GitHub Copilot Agent, ...]

Mensch → Ziel(prompt)



Agent plant



Tool aufrufen ← Shell / Dateien
 / Web / Tests



Ergebnis lesen



weiter iterieren



Ziel erreicht?

- Ein Ziel, viele autonome Schritte
- LLM **handelt**: schreibt, führt aus, liest
- Mensch gibt Richtung – Agent arbeitet
- Kontext bleibt über den ganzen Prozess erhalten

→ **Kontext Engineering** = Kunst, den Agenten mit dem richtigen Kontext zu versorgen



Vibe Coding

Karpathy, Feb 2025

*„Fully give in to the vibes, embrace exponentials,
forget that the code even exists.“*

- Code generieren → **Accept All** → hoffen
 - Kein Review, keine Tests, kein Verständnis
 - Optimiert für: schnelle Wegwerfprojekte
 - Verantwortung: **keine**
- Funktioniert für Prototypen und Wegwerfcode – **nicht** für Software, die jemand warten, deployen oder auf die sich jemand verlassen muss.

🎮 Beispiel: Ein Minigame per Vibe Coding



HU-Drop – das “HU”-Minispiel für Vibe-Coding-Experten

Wörter fallen von oben – klicke alle, die „hu“ enthalten.

Core Loop

- Wort erscheint 2 s an zufälliger Position
- Enthält „hu“ → **+1 Punkt** ✓
- Kein „hu“ → **-1 Punkt** ✗
- Steigende Spawn-Rate für mehr Spannung

Prompt → Accept All → läuft im Browser.

Beispiel: Der Prompt

Build a mobile-first web mini-game called "HU-Drop" (single-page app).

Core gameplay:

- Words spawn at the top of the screen and move downward (falling). X-position should be random within the playfield (avoid overlapping if easy).
- Spawn happens in a rapid cadence (start ~500ms; optionally ramp faster over time, but keep it simple if needed).
- Each word is visible/active for exactly 2.0 seconds from spawn. During those 2 seconds it may fall and/or slightly fade out near the end. After 2 seconds it disappears automatically.
- The player can tap/click a word while it's active.

Scoring:

- If the clicked word contains the substring "hu" case-insensitive (`word.toLowerCase().includes("hu")`), award +1 point.
- Otherwise apply -1 point.
- A word can only be clicked once; after click it should be removed immediately.
- Missing a "hu" word (not clicked before it expires) should NOT penalize (0 points).

UI:

- Top bar HUD: Left/center: Score display ("Score: X"). Right: Countdown timer ("Time: Y").
- Game area below HUD where words fall.
- End screen when timer reaches 0: Show final score. Buttons: "Play again" (restarts) and optional "Share" (can be stubbed).

Game timing:

- Round duration: configurable, default 20 seconds (acceptable range 15–30).
- Timer counts down in whole seconds; game stops spawning when time hits 0 and clears remaining words.

Content:

- HU-words: must contain "hu" (e.g., "Hupe", "Schuh", "Kuh", "Humor", "Hummel", "Husten", "HU").
- Neutral words: must NOT contain "hu".
- Spawn selection: target ~30% HU-words, ~70% neutral.

Implementation requirements:

- Use plain HTML/CSS/JS or a minimal framework (your choice), but keep it lightweight.
- Handle both mouse and touch reliably; word hit targets should be comfortably tappable (~44px height).
- Ensure smooth animation (`requestAnimationFrame` or CSS transitions).
- Keep logic clean and well-structured: game state, spawn scheduler, active word objects, click handler, timer loop, restart.

Deliverables:

- Provide a runnable project (single HTML file is fine) with clear instructions to run locally.
- Include comments explaining key parts: spawn logic, lifetime removal, scoring check, timer/end-state.

„Seasoned professionals accelerate their work – proudly accountable for the software they ship.“

- Agent = **LLM + Tools + Loop** → Ziel erreichen
- Code wird ausgeführt, getestet, iteriert
- Expertise wird verstärkt, nicht ersetzt
- Verantwortung: **bleibt beim Entwickler**

Die Teilkosten fürs Tippen sinken – die Kosten für *guten* Code nicht.

Korrektheit, Tests, Sicherheit, Wartbarkeit: weiterhin Aufgabe des Entwicklers.

Das Ökosystem: Coding Agents & CLIs

Terminal / CLI

- **Claude Code** – Sonnet 4.6, Opus 4.6
- **OpenAI Codex CLI** – GPT-5.2, GPT-5.3-Codex
- **Gemini CLI** – Gemini 3.1 Pro (1M Tokens)

IDE-Erweiterungen

- **GitHub Copilot Agent** – GPT-5, Claude Sonnet
- **Cline / Roo Code** – VS Code, beliebige Modelle
- **Continue** – open source, lokal oder cloud

AI-first IDEs

- **Cursor** – Sonnet, GPT-4.1, eigene Modelle
- **Windsurf (Codeium)** – SWE-1, Claude, GPT
- **VS Code + Copilot** – Edits-Modus, Agent-Modus

Cloud / Async Agents

- **Devin (Cognition)** – vollständig autonom
- **Jules (Google)** – Hintergrund-Tasks, GitHub-Integration
- **GitHub Copilot Workspace** – issue → PR
- **SWE-agent** – Forschung, SWE-Bench

Skills



Skills – Wissen on demand

Wie Neo, der in Sekunden Kung-Fu lernt:
Agenten bekommen Fähigkeiten *injiziert* –
nicht durch Training, sondern durch Kontext.

Was sind Skills?

Was sind Skills?

- Wiederverwendbare Prompt-Bausteine / Anweisungen
- Domänenwissen, Patterns, Coding-Standards
- Tool-Nutzungsanleitungen für den Agenten
- In Claude Code: /skill-Dateien im Projekt

Warum wichtig?

- Agent kennt *deine* Konventionen, nicht nur allgemeines Wissen
- Qualität steigt ohne den Agenten neu zu trainieren
- Skills = das neue Institutional Knowledge

Model Context Protocol

Was ist MCP?

Ein offenes Protokoll, das Agenten standardisiert mit externen Tools, Daten und Diensten verbindet.

„USB-C für KI-Agenten“

Das Problem davor:

- Jede App × jedes Modell = eigene Integration
- N×M Custom-Connectoren, kaum wiederverwendbar

Die drei Kern-Primitive:

 **Tools** – Aktionen, die der Agent ausführen kann

z.B. Datei lesen, API aufrufen, DB abfragen

 **Resources** – Daten, die der Agent lesen kann

z.B. Dateisystem, Dokumentation, Codebase

 **Prompts** – Wiederverwendbare Prompt-Templates

z.B. vordefinierte Workflows, Kontext-Snippets

- Anthropic, Nov 2024

Heute:

OpenAI, Google, alle großen Anbieter unterstützen MCP.

Seit Dez 2025 unter der Linux Foundation (AAIF).

Architektur:

Agent (MCP Client)

↔ MCP Server A (GitHub, Jira, ...)
↔ MCP Server B (Dateisystem, DB, ...)
↔ MCP Server C (eigene Tools, ...)

Appendix



Der Pelikan-Benchmark

Was wird gemessen?

Simon Willison bittet jedes Modell, per Prompt ein SVG zu erzeugen:

„Generate an SVG of a pelican riding a bicycle.“

Warum das interessant ist:

- Kein Trainingsdatensatz enthält viele Pelikan-auf-Fahrrad-SVGs
- Räumliches Denken + Code-Generierung kombiniert

Was es misst:

- Räumliches & physikalisches Verständnis
- Fähigkeit, *ausführbaren* Code zu erzeugen
- Kreativität vs. mechanische Korrektheit
- Fortschritt über Modellgenerationen hinweg

Ursprung:

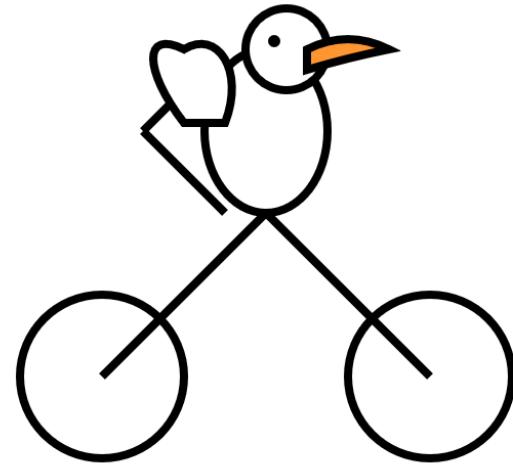
Simon Willison, 2024 —

- Fahrrad korrekt zu zeichnen ist überraschend schwer
- Ergebnis sofort visuell verständlich – kein Rubric nötig

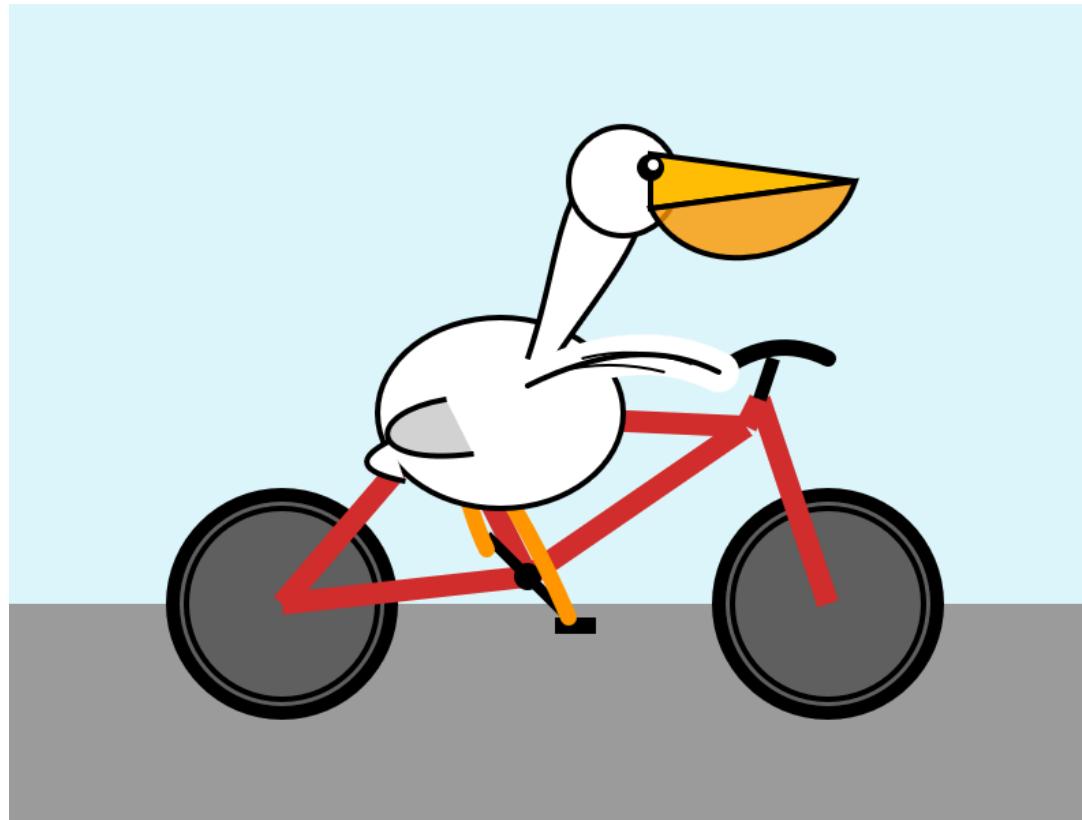
*„I started it as a joke,
but it's actually starting to be a bit useful.“*

github.com/simonw/pelican-bicycle

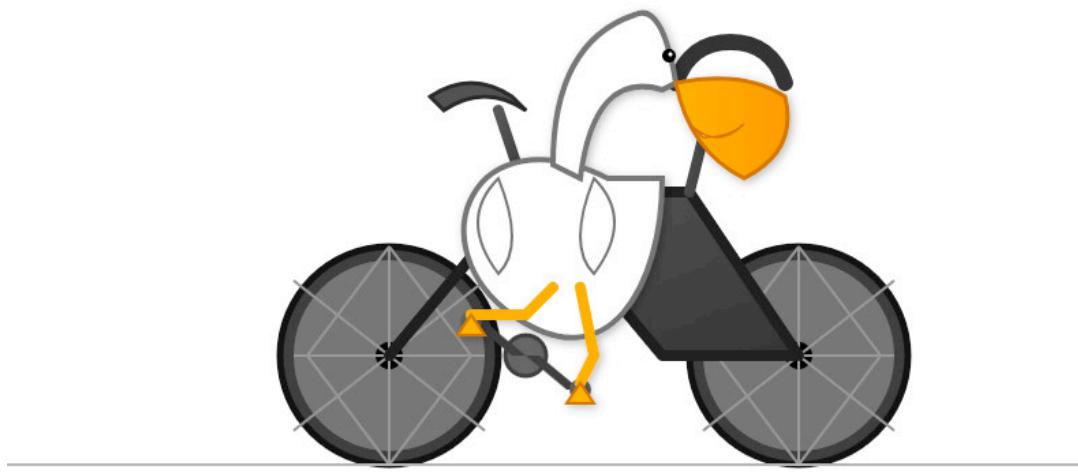
o1 Pro (High)



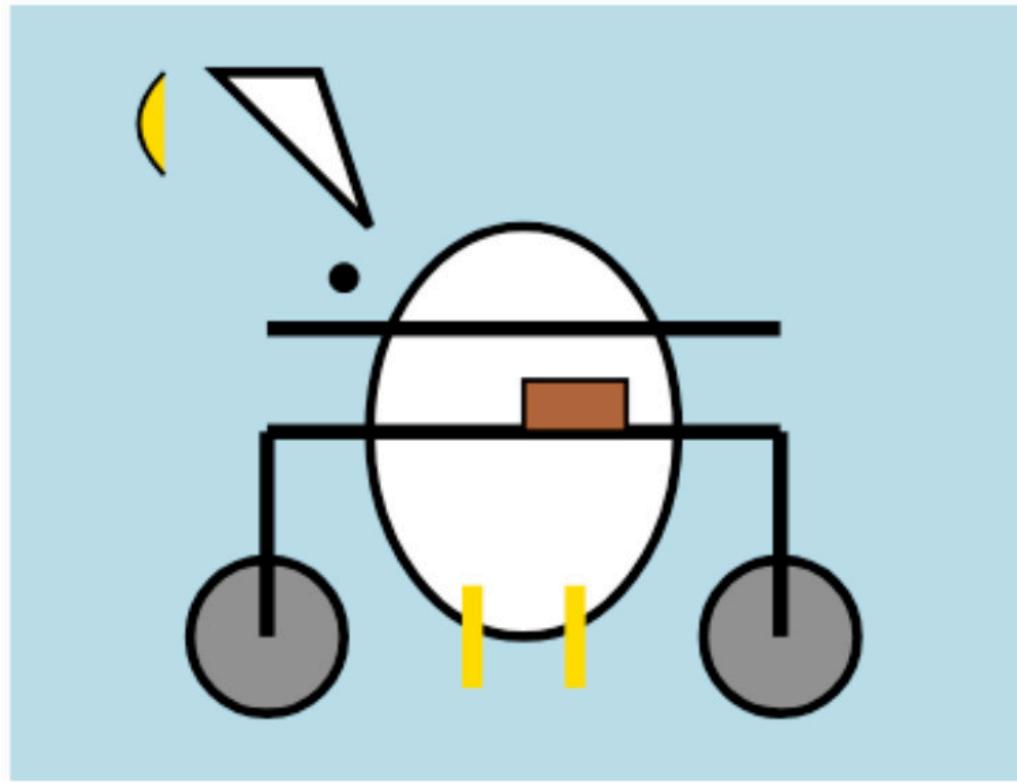
Deep Think



Gemini Flash Thinking



Gemma 3



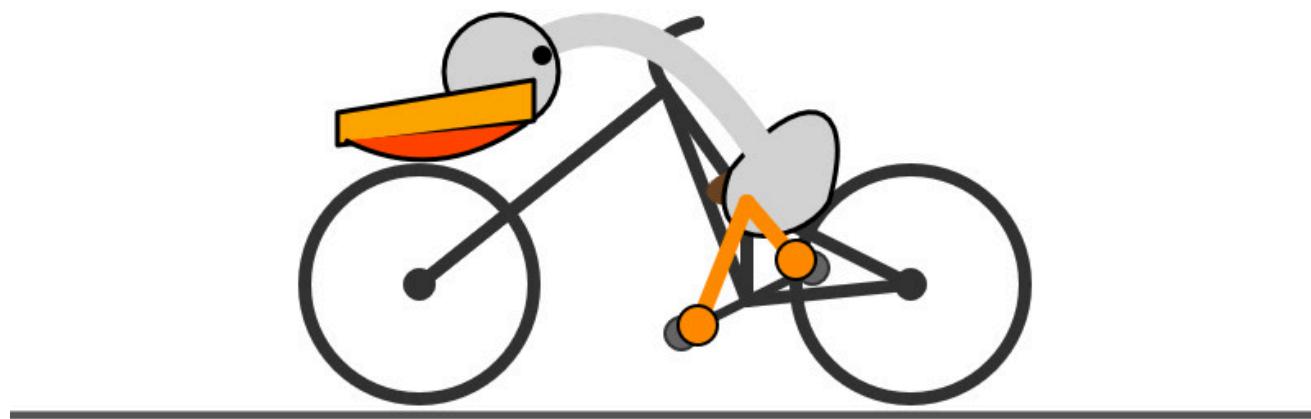
GPT-4.1



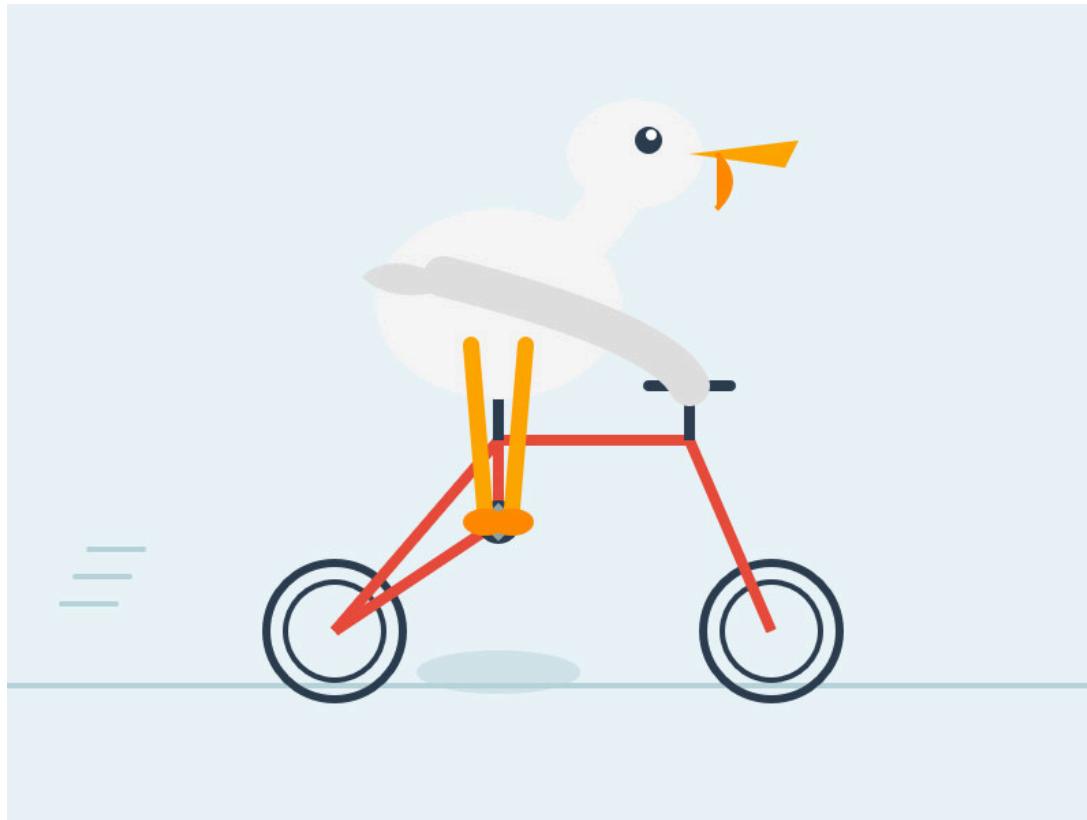
Gemini (latest)



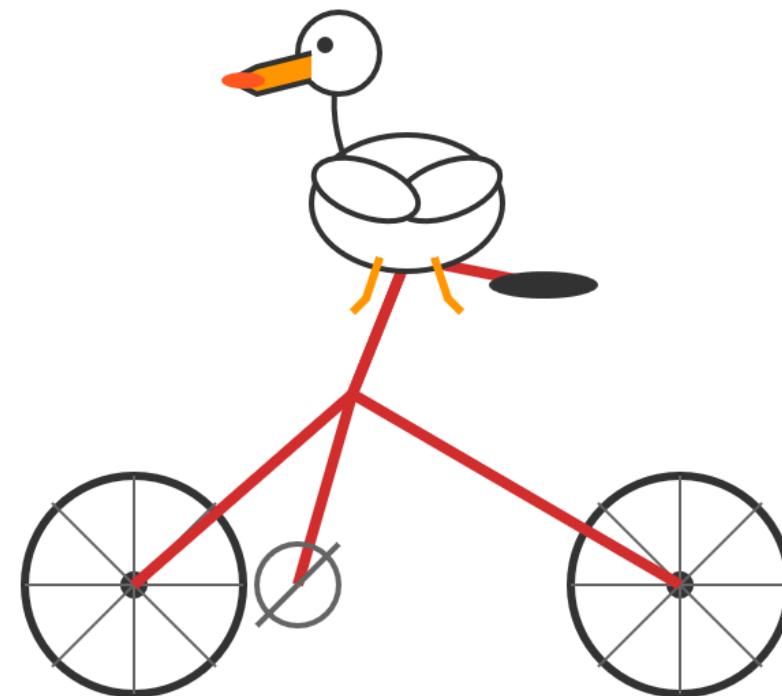
Gemini 2.5 Flash Thinking Max



GLM-4.5



Kimi K2



GPT-5



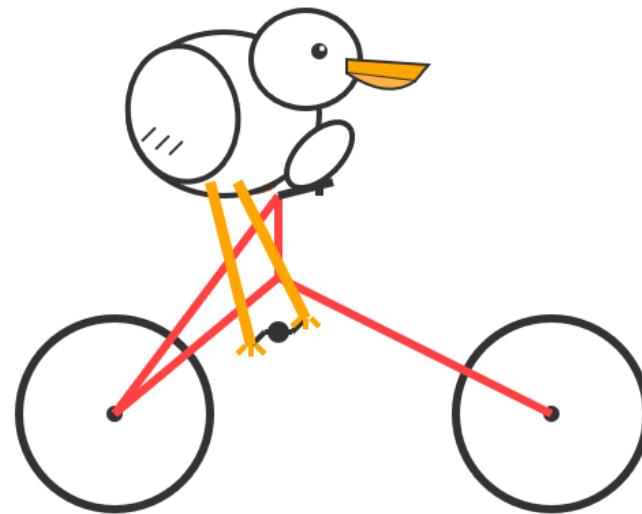
GPT-5 Codex API



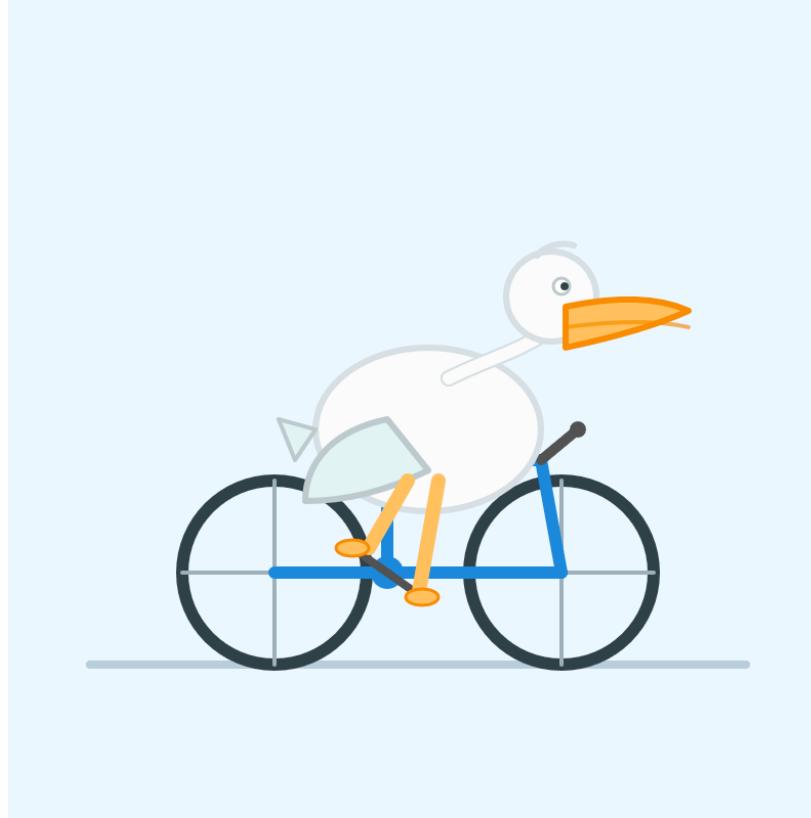
GPT-5 Pro



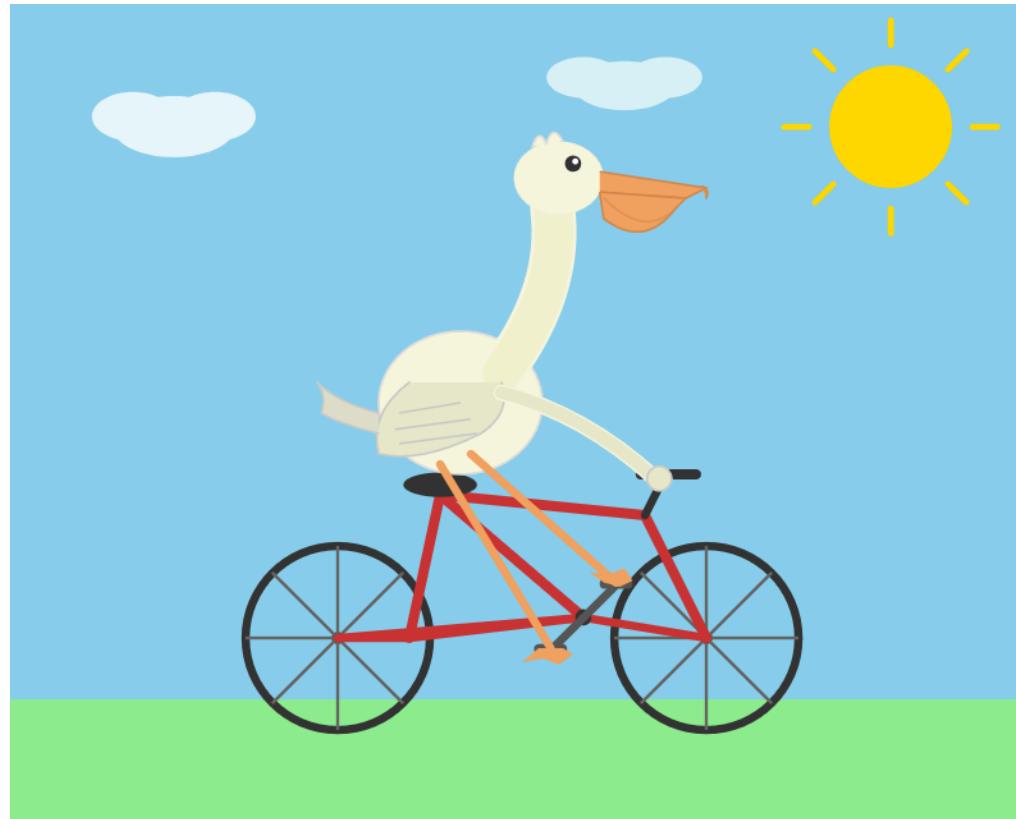
Claude Opus 4.1



GPT-5.1 High



Claude Opus 4.6



Gemini 3 Deep Think



Gemini 3.1 Pro

