Quick Start: Build Your First Self-Improving System

From Zero to Working System in 30 Minutes

By Mark Kashef

Table of Contents

Introduction

This guide gets you from zero to a working self-improving AI system as fast as possible. You'll have a chatbot that evaluates its own responses and updates its system prompt automatically.

Related guides:

- *Architecture Guide* - Deep dive into system design
- *Rubric Template* - Customize evaluation criteria
- *Handoff Template* - Manage context across sessions

Prerequisites

Before starting, ensure you have:

Requirement	How to Check	How to Install
Claude Code CLI	`claude --version`	*claude.ai/cli*
Supabase account	Login at supabase.com	*supabase.com*
Node.js 18+	`node --version`	*nodejs.org*
Anthropic API key	Have it ready	*console.anthropic.com*

Time estimate: 30-45 minutes for first-time setup, 15 minutes if you've used Supabase before.

Step 1: Enable Supabase MCP Server

The MCP (Model Context Protocol) server lets Claude Code interact directly with your Supabase database.

1.1 Create a Supabase Project

Go to *supabase.com/dashboard*
Click "New Project"
Choose your organization
Enter project details:

- **Name:** `self-improving-ai` (or your preference)

    - **Database Password:** Generate a strong password (save it!)
    - **Region:** Choose closest to you

Click "Create new project"

Wait 2-3 minutes for provisioning

1.2 Get Your Credentials

From your Supabase project dashboard:

Click "Settings" (gear icon) → "API"

Copy these values:

- **Project URL:** `https://xxxxx.supabase.co`
- **anon public key:** `eyJhbG...` (under "Project API Keys")
- **service_role key:** `eyJhbG...` (click "Reveal" - keep this secret!)

1.3 Configure Claude Code MCP

In your terminal, run:

```
claude mcp add supabase
```

Follow the prompts to enter:

- Your Supabase project URL
- Your service_role key (for full database access)

Verify it's working:

```
claude mcp list
```

You should see `supabase` in the list of configured servers.

1.4 Test the Connection

Start Claude Code and test:

```
List all tables in my Supabase database.
```

Expected response: "There are no tables yet" or a list if you have existing tables.

Step 2: Use the Mega Prompt

The mega prompt tells Claude Code to build the entire self-improving system. It's your one-shot instruction set.

2.1 Create Your Project Directory

```
mkdir self-improving-ai
cd self-improving-ai
```

2.2 Start Claude Code

```
claude
```

2.3 Send the Mega Prompt

Copy and send this entire prompt:

```
Build a self-improving AI chatbot with these components:

## Database Schema (Create in Supabase via MCP)
1. users table: id (uuid pk), email (text unique), created_at
2. sessions table: id (uuid pk), user_id (fk), title, created_at, updated_at, is_active
3. messages table: id (uuid pk), session_id (fk), role (user/assistant/system), content, created_at,
4. system_prompts table: id (uuid pk), version (int), content (text), is_active (bool), created_at, c
5. reflection_logs table: id (uuid pk), created_at, messages_analyzed (int), prompt_version_evaluated

## Edge Functions (Create in supabase/functions/)

### chat-handler
- Receives: { message, session_id }
- Gets active system prompt from system_prompts where is_active = true
- Gets conversation history from messages
- Calls Claude API with system prompt + history + new message
- Stores user message and assistant response with current prompt_version
- Returns assistant response

### reflection-loop
- Query last 20 assistant messages with their user messages and prompt versions
- Use Claude to evaluate responses against this rubric:
  - Completeness (1-5): Did response address all parts of question?
  - Depth (1-5): Did response provide sufficient detail?
  - Tone (1-5): Was communication style appropriate?
  - Overall = weighted average
- If overall &lt; 2.5: auto-generate improved prompt and update system_prompts
- Log everything to reflection_logs

## Frontend (Next.js in /frontend)
- Simple chat interface
- Session sidebar (list, create, switch)
- Messages list with streaming
- Input form

## Initial System Prompt (insert as version 1, is_active = true)
&quot;You are a helpful AI consultant specializing in strategy and problem-solving.
Be concise but thorough. Ask clarifying questions when needed.
Provide actionable advice, not just information.&quot;

Start by creating the database schema, then the edge functions, then a minimal frontend.
After each component, verify it works before moving on.
```

2.4 Let Claude Code Build

Claude Code will now:

- Create all database tables via MCP
- Generate edge function code
- Create a frontend
- Set up the initial system prompt

This takes 5-15 minutes depending on how much Claude Code explains along the way.

Tip: If Claude Code asks questions, answer them. If it gets stuck, say "continue" or describe the specific issue.

Step 3: Test the Basic Chat Flow

3.1 Set Up Edge Function Secrets

Your edge functions need the Anthropic API key. In Supabase:

- Go to your project dashboard
- Click "Edge Functions" in the sidebar
- Click "Manage secrets"
- Add secret:
  - **Name:** `ANTHROPIC`*`API`*`KEY`
  - **Value:** Your Anthropic API key

3.2 Deploy Edge Functions

In Claude Code:

```
Deploy the chat-handler and reflection-loop edge functions to Supabase.
```

Or manually:

```
cd supabase/functions
supabase functions deploy chat-handler
supabase functions deploy reflection-loop
```

3.3 Start the Frontend

```
cd frontend
npm install
npm run dev
```

Open *http://localhost:3000*

3.4 Test Chat

Create a new session

Send a message: "What's the best way to prioritize tasks?"

Verify you get a response

Check Supabase: messages table should have 2 new rows

Troubleshooting:

- No response? Check edge function logs in Supabase
- CORS error? Verify edge function URL is correct
- Auth error? Check API keys are set correctly

Step 4: Trigger Your First Reflection

4.1 Generate Test Data

Have a few conversations with your chatbot. You need at least 5-10 assistant messages for meaningful reflection.

Quick test prompts:

- "How do I stay focused while working from home?"
- "What makes a good morning routine?"
- "How should I prepare for a difficult conversation?"
- "What's the 80/20 rule and how do I apply it?"

4.2 Trigger Reflection Manually

In Claude Code:

```
Call the reflection-loop edge function and show me the results.
```

Or via curl:

```
curl -X POST https://YOUR_PROJECT.supabase.co/functions/v1/reflection-loop \
  -H &quot;Authorization: Bearer YOUR_ANON_KEY&quot; \
  -H &quot;Content-Type: application/json&quot;
```

4.3 Check the Reflection Log

In Claude Code:

```
Show me the latest entry in reflection_logs.
```

You should see:

- Scores for each criterion
- Strengths and weaknesses identified
- Action taken (none, suggestion, or prompt_update)

Example output:

```
{
  &quot;overall_score&quot;: 3.8,
  &quot;completeness_score&quot;: 4,
  &quot;depth_score&quot;: 3,
  &quot;tone_score&quot;: 4,
  &quot;strengths&quot;: [&quot;Clear and actionable advice&quot;, &quot;Good structure&quot;],
  &quot;weaknesses&quot;: [&quot;Could provide more specific examples&quot;],
  &quot;action_taken&quot;: &quot;suggestion&quot;
}
```

Step 5: Verify the Feedback Loop Works

5.1 Force a Low Score (Optional Test)

To test the auto-update mechanism, temporarily lower the threshold:

```
-- In Supabase SQL editor, or ask Claude Code
UPDATE system_prompts SET content = 'You are an AI. Answer questions.' WHERE is_active = true;
```

This weak prompt should produce lower-quality responses.

5.2 Generate New Messages

Chat a few more times with the weakened prompt:

```
&quot;How do I negotiate a raise?&quot;
&quot;What's the best approach to learning a new skill?&quot;
```

5.3 Run Reflection

Trigger another reflection:

```
curl -X POST https://YOUR_PROJECT.supabase.co/functions/v1/reflection-loop \
  -H &quot;Authorization: Bearer YOUR_ANON_KEY&quot;
```

5.4 Verify Prompt Update

Check if a new prompt version was created:

```
SELECT version, content, created_at, created_by
FROM system_prompts
ORDER BY version DESC
LIMIT 2;
```

You should see:

- Version 2 (or higher) created by 'system'
- New content that addresses the weaknesses

5.5 Test Improved Responses

Chat again with the same questions. Responses should be noticeably better now that the system improved its own prompt.

Common Errors and How to Fix Them

Error: "Function not found"

Symptom: 404 when calling edge functions

Fix:

- Verify function is deployed: `supabase functions list`
- Check function name matches URL path
- Redeploy: `supabase functions deploy function-name`

Error: "Invalid API key"

Symptom: 401 or authentication error

Fix:

- Check ANTHROPIC*API*KEY is set in Supabase secrets
- Verify key is valid at console.anthropic.com
- Restart edge function (redeploy triggers restart)

Error: "No messages to analyze"

Symptom: Reflection returns empty or errors

Fix:

- Ensure messages exist in database
- Check time window—reflection looks at recent messages
- Verify messages have `prompt_version` populated

Error: "CORS policy"

Symptom: Browser blocks API calls

Fix:

Add CORS headers to edge functions:

```
const corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'authorization, x-client-info, apikey, content-type',
};

// In your function
if (req.method === 'OPTIONS') {
  return new Response('ok', { headers: corsHeaders });
}

// Add to all responses
return new Response(JSON.stringify(data), {
  headers: { ...corsHeaders, 'Content-Type': 'application/json' },
});
```

Error: "Rate limit exceeded"

Symptom: 429 error from Anthropic

Fix:

- Add delay between API calls
- Reduce reflection frequency
- Check Anthropic usage dashboard

Error: "Prompt version null"

Symptom: Messages saved without prompt_version

Fix:

Ensure chat-handler includes prompt version:

```
// When inserting messages
await supabase.from('messages').insert({
  session_id,
  role: 'assistant',
  content: response,
  prompt_version: currentPrompt.version  // Don't forget this!
});
```

What You've Built

Congratulations! You now have:

Component	What It Does
Chat interface	Users interact with your AI
Message persistence	All conversations saved with version tracking
Reflection loop	AI evaluates its own performance
Auto-update	System improves its own prompts
Version history	Full audit trail of all changes

The Loop in Action

```
User chats → Messages saved → Reflection runs →
Scores calculated → Prompt updated → Better responses →
User chats → ...
```

Next Steps

Immediate Improvements

Add a scheduler: Set up cron to run reflection every 12 hours

Build admin panel: View reflection logs and prompt history

Customize rubric: Adjust criteria weights for your use case

Advanced Features

A/B testing: Run two prompts simultaneously, compare scores

Human-in-the-loop: Require approval for low-confidence updates

Multi-domain: Different prompts for different conversation types

Learn More

- *Architecture Guide* - Understand the system deeply
- *Rubric Template* - Customize evaluation criteria
- *Handoff Template* - Manage context across sessions

Quick Reference Card

```
# Start frontend
cd frontend &amp;&amp; npm run dev

# Deploy edge function
supabase functions deploy chat-handler

# Trigger reflection
curl -X POST https://PROJECT.supabase.co/functions/v1/reflection-loop \
  -H &quot;Authorization: Bearer ANON_KEY&quot;

# Check latest reflection
SELECT * FROM reflection_logs ORDER BY created_at DESC LIMIT 1;

# Check prompt versions
```

```
SELECT version, is_active, created_by, created_at
FROM system_prompts ORDER BY version DESC;

# Rollback prompt
UPDATE system_prompts SET is_active = false WHERE is_active = true;
UPDATE system_prompts SET is_active = true WHERE version = N;
```

This quick start is part of the Self-Improving AI Systems package. See the Architecture Guide for a deeper understanding.