# Jaemyeong Jin

**iOS Software Engineer · Personal Project Portfolio**

jaemyeong@me.com · 010-3614-3752 · jaemyeong.com · github.com/jaemyeong

## Project Overview

The four projects below are personal PoC projects I built to learn new technologies by using them hands-on.

### DailySudoku | Sudoku app for iPhone and iPad

pp. 2–3

Started 2026.04 · Released on the App Store · dailysudoku.app

A sudoku app with a new puzzle every day. Covers handwritten number input and how the time shown on screen was kept consistent with the time that is saved.

**Technologies explored:** Rust shared engine (UniFFI), Core ML on-device inference, Swift 6

---

### Ahob | Block puzzle for iPhone and iPad

p. 4

Released 2020.11 · First Prize, 5th Bu:Star Challenge Game Contest · ahob.app

A 9×9 block puzzle built with UIKit alone, without a game engine. Covers the game screen implemented in UIKit and the work on accessibility and testing.

**Technologies explored:** UIKit-based game rendering and animation, accessibility (VoiceOver), Game Center

---

### NutritionEngine | Food nutrition analysis service

pp. 5–6

Started 2026.06 · nutritionengine.io

A service that returns nutrition information for a food name or photo. Covers the architecture that keeps AI from making up numbers and how the search method was chosen through experiments.

**Technologies explored:** RAG, local LLM serving and quantization, pgvector, MCP

---

### StorageGame | Puzzle game for Mac

p. 7

Started 2026.08 · Released on the Mac App Store and itch.io · storage-game.com

A game that fits within the size of a single floppy disk. Covers how it was made without image or sound files and how its configuration was split by distribution channel.

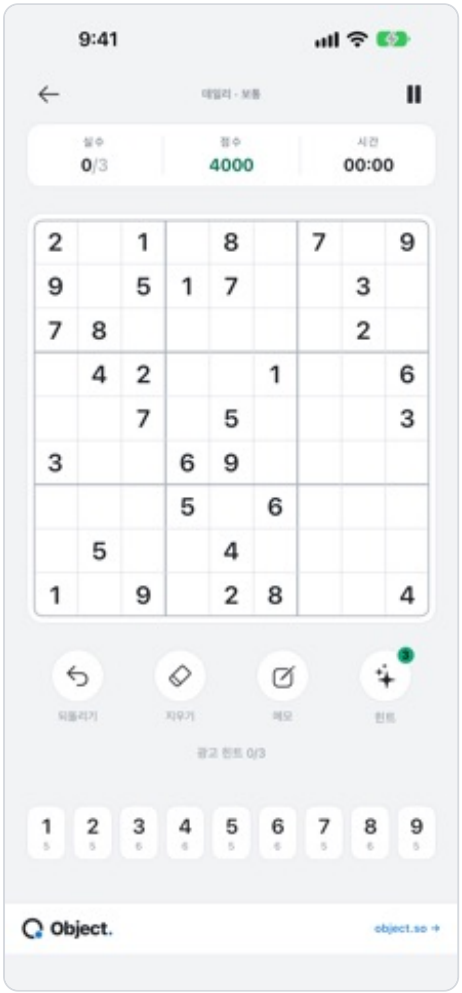
**Technologies explored:** SpriteKit, code-drawn graphics and synthesized sound, binary size optimization

DailySudoku · iPhone and iPad app

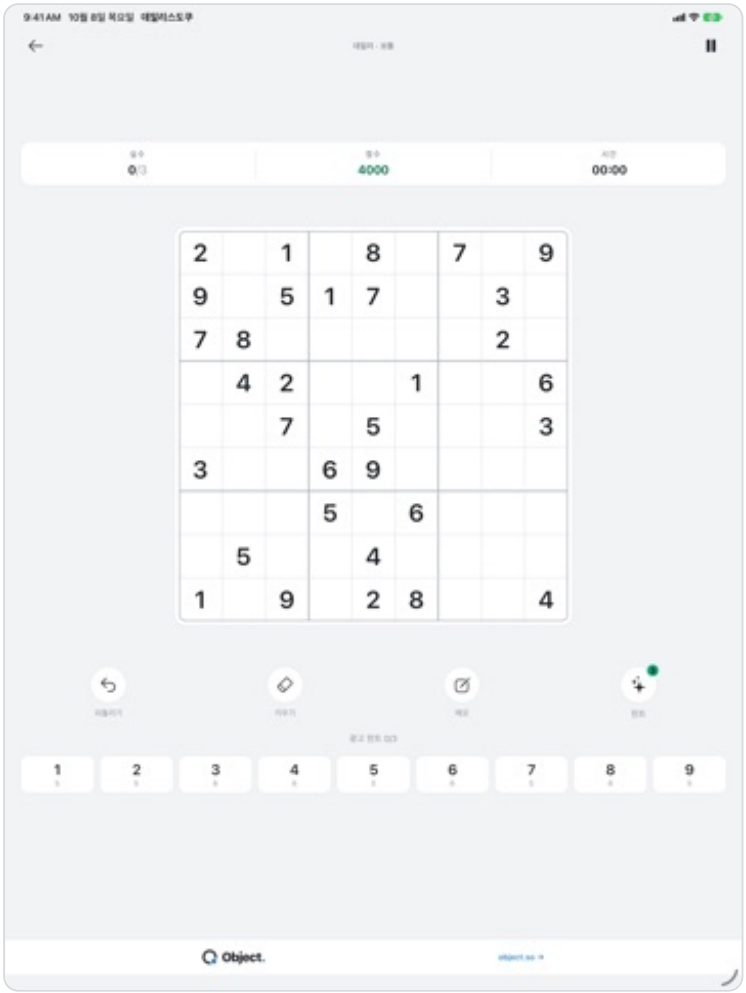
# A sudoku app for solving a new puzzle every day and keeping your record going

Started 2026.04 · Released on the App Store · dailysudoku.app

In DailySudoku, you pick a date and difficulty to get that day's puzzle, then solve it with numbers and notes. I started it to validate firsthand an architecture that manages, in one place, the parts that must behave identically on every device, such as the game rules. I handled everything from planning through design, implementation, and release myself.




Game screen on iPhone




Game screen on iPad

# A sudoku app for solving a new puzzle every day and keeping your record going

## Handwritten number input

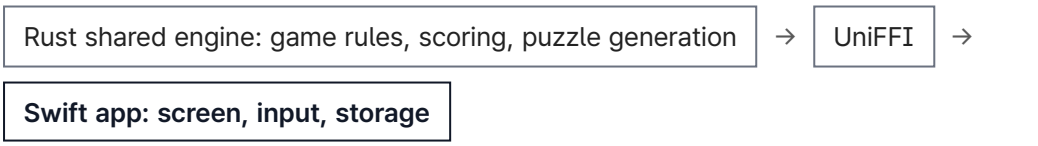
Writing a number in a cell with Apple Pencil enters it immediately. To recognize handwriting on the device without going through a server, I embedded a small recognition model in the app. (Core ML)

- I built tests that check whether the training environment and the app recognize the same handwriting as the same digit, so that differences in input preprocessing cannot change the result.

## Keeping on-screen time and saved time in sync

The score changes every second with elapsed time. Instead of recomputing the entire game state every second, I changed the displayed score to be computed from only the four numbers it needs (base score, elapsed time, mistakes, hints). Because this could let the on-screen time drift from the saved time, the accumulated time is written into the game state right before a number is entered or the game is saved or completed.

- The first test checked only the on-screen time, so it still passed with the pre-save update removed. I fixed the test to check the saved time.
- I split responsibilities so that the Rust shared engine owns the game rules and scoring while the app handles the screen, input, and storage. (Rust shared engine connected to the Swift app)




## Technical summary

- Connected the Rust shared engine to the Swift app with UniFFI. The same date and difficulty generate the same puzzle regardless of platform.
- Implemented a bitmask/MRV-based unique-solution solver and a SplitMix64 random number generator myself.
- Handwriting recognition runs an MNIST CNN (about 110,000 parameters, about 215KB in fp16) on device with Core ML.
- Adopted the Swift 6 language mode and automated releases with GitHub Actions, Xcode Cloud, and fastlane.

Technologies used: Swift · UIKit · Core ML · Rust (UniFFI)

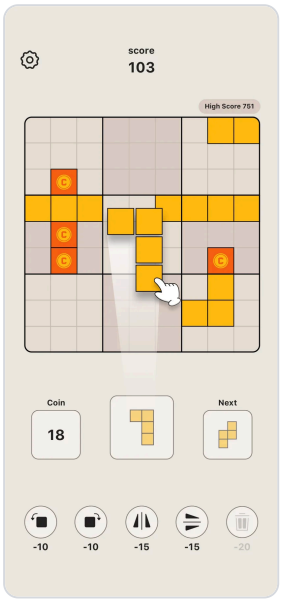
Ahob · iPhone and iPad game

# A 9×9 block puzzle built with UIKit alone, without a game engine

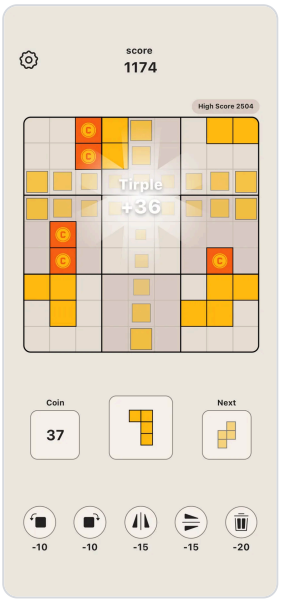
Released 2020.11 · First Prize, 5th Bu:Star Challenge Game Contest · ahob.app

---

A puzzle game in which placing blocks to fill a row, a column, or a 3×3 region clears it. I started it to understand rendering and animation in depth by building an interactive game with UIKit alone, without a game engine, and to go through the entire process from planning to release myself.




Game screen




Combo

## A game built with UIKit

- Implemented the board, block dragging, and animations in UIKit without a game engine. Gameplay works offline.
- Supports 3 Game Center leaderboards, an in-app purchase to remove ads, the UMP consent flow, 40 locales, left-handed mode, and dark mode.

## Accessibility and testing

- In the 2026 overhaul, carried out after leaving the company, I compared the design canvas one-to-one against 38 screenshots of the actual screens.
- Within the audited scope, reduced duplicate VoiceOver exposure and tap targets under 44pt to zero, and increased the number of tests from 374 to 509.

## Stages verified by algorithm

- Proved the minimum number of moves for 9 campaign stages with BFS search.

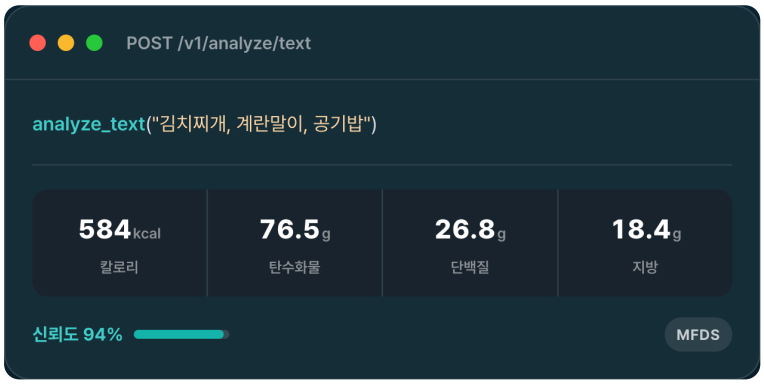
Technologies used: Swift · UIKit · Game Center · StoreKit

NutritionEngine · Food nutrition analysis service

# A service that returns sourced nutrition information for a food name or photo

Started 2026.06 · nutritionengine.io

NutritionEngine is a service that other apps and AI tools use to look up nutrition information for foods. Send a sentence describing a food, or a photo, and it returns the nutrients. I built it to see for myself whether the problem of AI making up plausible numbers could be prevented by architecture.




Example analysis request and result

## Separating the roles of AI and data

The AI (language model) decides only what the food is, how much of it there is, and which search terms to use. Nutrient values come only from public food data. The AI's output format has no field for nutrient values at all.

Food text or photo → AI: food, portion, search plan → Food data search → Source and value check → **Nutrition result**

- Food data from the Ministry of Food and Drug Safety (MFDS), USDA, and Open Food Facts is normalized and linked, and searched with BGE-M3 embeddings and a pgvector HNSW index.
- For the language model, I served a quantized local model (Gemma 4 12B QAT) myself and connected it. (LM Studio, Python, FastAPI)

## Checking that values were not changed along the way

A retrieved value can be altered, or lose its source, in later processing steps. So I keep a separate copy of the values right after retrieval and compare against it before returning the final result.

- During review, I found and closed paths that could slip past this comparison. For example, Python treats True and the number 1 as equal, so a value changed to True could pass the comparison. I fixed it to compare the type as well as the value.

# Choosing search by experiment and assigning implementation and review to AI tools

## How it is offered

- Offered as a REST API (called from apps), an MCP server (called from AI tools), and a console (for viewing analysis results and usage).
- Added a two-tier LLM response cache, and used Idempotency-Key so that usage is not recorded twice when the same request is retried.

## Search method comparison experiment

Users type food names without spaces, only in part, or with typos. I ran an experiment to decide which should be the default: search that finds similar meanings (dense), or a method that combines it with exact character matching (hybrid).

I picked 200 food names from the development data and built 904 queries covering removed or added spaces, typos, and prefix-only or partial input, then compared the two methods.

| Metric | Meaning-based (dense) | Hybrid |
|---|---|---|
| Correct answer ranked 1st | 53.2% | 54.8% |
| Correct answer in the top 3 | 66.3% | 69.7% |
| Correct answer in the top 20 | 77.4% | 81.6% |

Hybrid scored higher overall. But for queries with spaces removed, the correct answer was ranked 1st 83.6% of the time with dense and 74.5% with hybrid, so hybrid was actually lower. Judging that switching based on the average alone would worsen results for common inputs, I kept the default unchanged and offered hybrid as an optional setting.

- These results evaluate the search stage only. (317,483 rows of development data, BGE-M3 embeddings, RRF fusion)

## Splitting implementation and review across AI tools

I worked with AI tools to lock in the console's "loading, no results, error, and per-permission display" states with tests.

- First, I tabulated the screen states and wrote down which files could be changed, what was off-limits, and the completion criteria.
- I assigned implementation to Codex, which added 32 state tests, and had Claude review the same changes separately.
- After the tools reported they were done, I ran the full test suite myself to confirm. (47 files, 356 tests passed)

## What I learned

Checking whether the AI's output was correct was not enough. It was more reliable to limit, through architecture, the range of values the AI can change, and to close with tests the paths that could bypass verification.

Technologies used: Python · FastAPI · PostgreSQL · pgvector · TypeScript · Local LLM

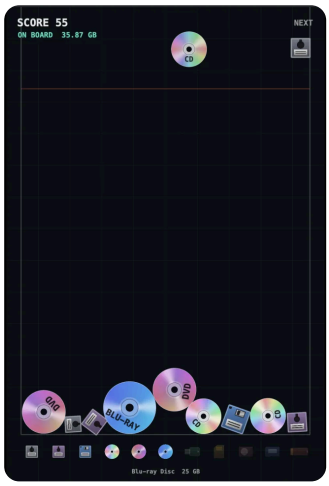
StorageGame · Mac game

# A puzzle game where you merge storage media into larger storage media

Started 2026.08 · Released on the Mac App Store and itch.io · storage-game.com

---

A Mac game in which you place blocks with the keyboard and merge identical storage media to grow them into higher-capacity media. I set the constraint of fitting the entire app within the size of a single floppy disk (1.44MB) and explored how to create graphics, sound, and networking within that limit.




Gameplay




Menu and merge stages

## Built without image or sound files

To keep the size down, I included no image or sound files other than the app icon. Graphics are drawn in code, and sound effects are computed when the app launches and then reused during play. (SpriteKit, AppKit, sine-wave synthesis)

## Different configurations per distribution channel

External libraries commonly used for online features are large, so in the directly distributed version I implemented only the networking features it needs myself. The Mac App Store version was split off to include the library conditionally. (REST and SSE on URLSession)

- Size is measured on the fully signed app, and the app itself is tracked separately from the distribution installer (DMG). The measured app bundle size is 775,426 bytes, about 53% of the limit (1,474,560 bytes).
- Built tools for building, creating the installer, and generating store screenshots so that release work is repeatable.
- Built as a universal binary for Apple Silicon and Intel.

Technologies used: SpriteKit · AppKit · Objective-C++ · C++ · URLSession