songgot · 송곳

PAPER · PALETTE · 2026-09-10

A Korean-first tiny agentic model, for tool calling on the device.

Songgot is a from-scratch tiny language model (tens of millions of parameters) for Korean tool calling and structured extraction on phones and small devices, with a Korean-first 32k tokenizer, licence-clean data, and reproducible exact-match evaluation on Kakao FunctionChat-Bench. Weights, code and paper under Apache 2.0.

Weights on Hugging Face | Code on GitHub | Live demo | Pocket app (offline, in the browser)

Paper as PDF | Paper as Markdown

0.90
tokens per Hangul syllable (Songgot 32k tokenizer); Needle 2: 3.47

500
Korean items, FunctionChat-Bench SingleCall, exact-match scorer

Apache 2.0
weights, tokenizer, code, data generators, paper

0 closed-model labels
licence-clean data, provenance in the paper

Hanish Keloth, Palette (Seoul and Bengaluru). Draft of 2026-09-10. Songgot rows in Table 2 are written from the run logs as each training run finishes; nothing on this page is estimated.

Abstract

Tiny agentic models (Needle 2, 45M parameters, 14 MB, July 2026) made on-device tool calling practical on phones and microcontrollers, but the released model is English only. Songgot is a

Korean-first model in the same class: a decoder trained from scratch with a Korean-first 32k tokenizer, bilingual Korean and English pretraining on licence-clean text, and post-training on Korean tool-calling and structured-extraction data that no closed model touched. We evaluate on Kakao's FunctionChat-Bench SingleCall (500 Korean items) with a deterministic exact-match scorer that anyone can reproduce without an API key, against Needle 2, FunctionGemma-270M, Qwen3-0.6B and Qwen3.5-0.8B under identical prompts. Results for the comparators are measured and in Table 2; Songgot rows are filled from the run logs as each training run finishes (Songgot-nano on 2026-09-10). Weights, tokenizer, data generator, scorer and this paper are released under Apache 2.0.

1. Why a Korean-first tiny model

- The class exists and is growing: Needle 2 passed 52,000 downloads within weeks of release.
- On Hugging Face as of 2026-09-09 there is no Korean tool-calling model under 1B parameters; the one Korean function-calling finetune found is a 2B Gemma.
- Korean is expensive for English-first tokenizers. On the 100 FunctionChat SingleCall queries, Qwen3's 151k vocabulary spends 1.15 tokens per Hangul syllable, Gemma's 262k vocabulary 0.98, and Songgot's 32k Korean-first vocabulary 0.90 (Table 1). At a 256 to 1024 token budget shared with tool schemas, that is context.

2. Model

Llama-style decoder, hidden 512, GQA with 8 query and 2 key-value heads, SwiGLU intermediate 1408, RoPE, tied embeddings, 32k vocabulary, trained from scratch. Two sizes: Songgot-nano, 8 layers, 39M parameters, pretrained on an Apple M5 Max with MLX (320M tokens); Songgot, 12 layers, about 50M parameters, 8xH100 run on 6B tokens queued behind GPU budget. Exports to safetensors, GGUF (f16, Q8_0, Q4_K_M) and runs under llama.cpp.

Figure 3. Songgot-nano pretraining loss on the Mac, read from the run log

3. Tokenizer

SentencePiece BPE, 32,000 pieces, byte fallback, digits split, trained on a 300 MB sample: Korean Wikipedia 50 percent, fineweb-edu 40 percent, tool-schema JSON 10 percent. Special tokens `<|system|> <|user|> <|call|> <|end|> <|pad|>`. Post-training, evaluation and the app share one tokenizer: llama.cpp's, driven through its Python bindings with a vocabulary-only GGUF (the five specials typed as user-defined tokens). SentencePiece folds newlines into spaces and inserts a space piece before each special; llama.cpp does neither, and it is what every GGUF user runs, so the model is trained and scored on those ids.

Table 1. Tokens per Hangul syllable on the 100 FunctionChat SingleCall queries (2,071 syllables).

TOKENIZER	VOCAB	TOKENS	TOKENS PER SYLLABLE
Songgot (ours)	32k	1,873	0.90
Gemma 3 / FunctionGemma	262k	2,028	0.98
Qwen3	151k	2,392	1.15
Needle 2	8k	7,192	3.47

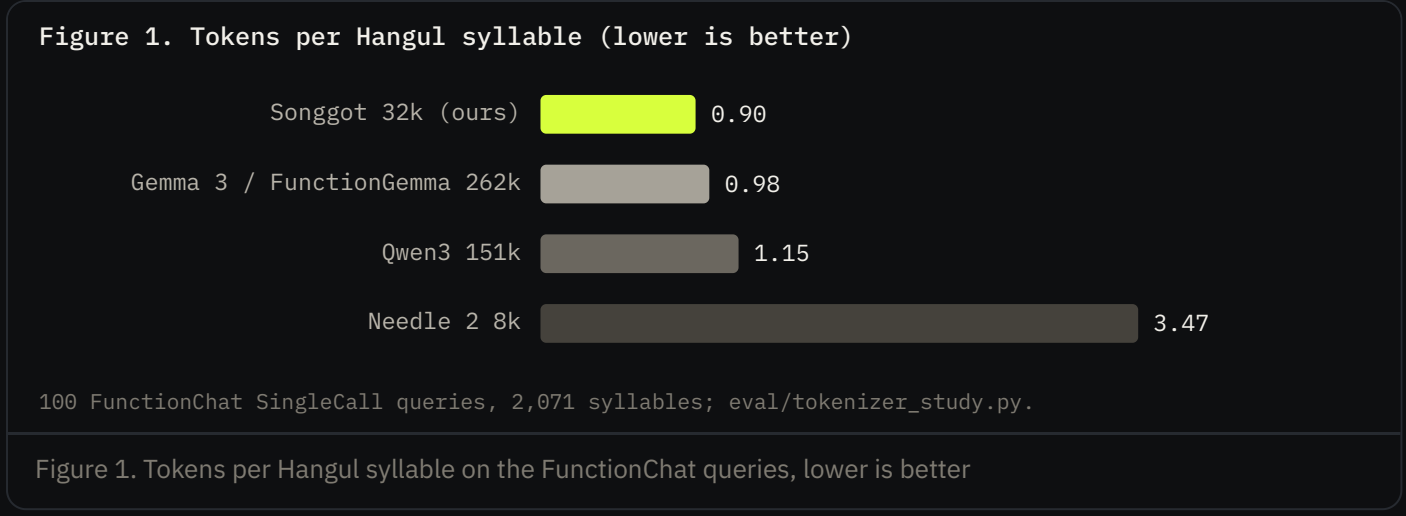

Figure 1. Tokens per Hangul syllable on the FunctionChat queries, lower is better

4. Data

Pretraining (all disclosed, all licence-clean):

- fineweb-edu sample-10BT (ODC-By), first 1.63B tokens under our tokenizer.
- Korean Wikipedia, dump 20231101.ko (CC BY-SA 3.0), 0.60B tokens, upsampled to roughly half of the training mix.
- No AI-Hub data (its terms forbid leaving Korea and our GPUs do not sit there), no crawled Korean web text of unclear licence.

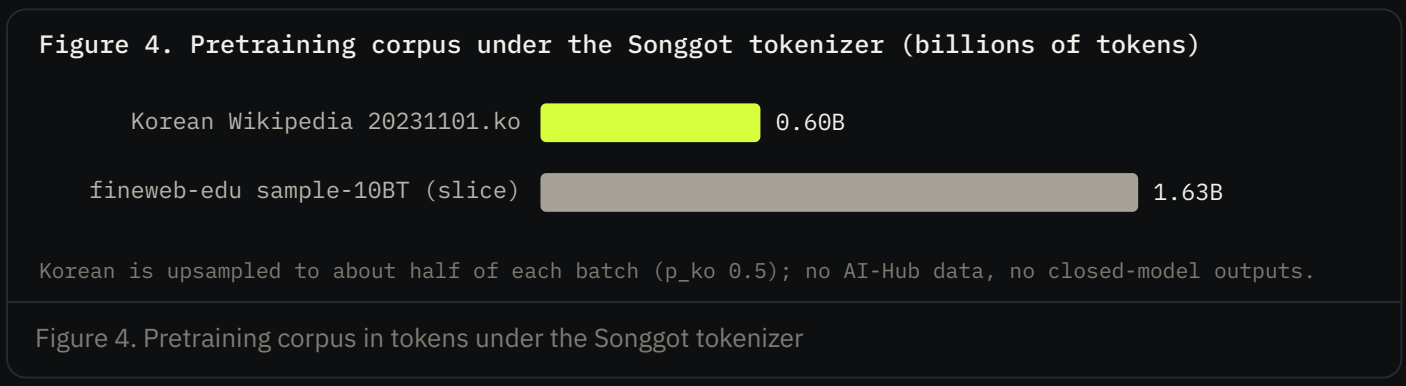

Figure 4. Pretraining corpus in tokens under the Songgot tokenizer

Post-training (101,464 examples, set v3): 64,000 Korean tool calls generated from hand-written frames for 57 Korean tools (messaging, calendar, transit, home appliances, commerce, public services, health, work, information), with register variants (반말, 존댓말, 격식, 사투리, typos) and Korean date and time words resolved against a fixed calendar; 8,083 Korean calls generated by our own Palette-K-Midm (open weights) as teacher; 27,948 English calls from glaive-function-calling-v2 (Apache 2.0) over thousands of distinct tools; 6 percent "no tool applies" negatives. Each prompt carries 1, 3 to 6 or 8 tools; half of the multi-tool Korean prompts use close distractors (tools from the target's own category), mirroring the benchmark's conditions.

Tool-name style augmentation, the change that mattered most: in 52,165 rows every tool is renamed in a random style (camelCase, PascalCase, snake_case, digit or word suffixes, verb synonyms; 47,192 distinct names in the set) and the target call is renamed to match. Our first 12-layer model, trained without it, scored 0.2 percent: it copied any invented snake_case name

exactly and could not copy a single camelCase one, because every name it had seen was snake_case (the benchmark mixes both). A tiny model learns the name style it is shown; showing it every style teaches it to copy the name from the prompt. Two related fixes: glaive's arguments are single-quoted and 97 percent of its calls had failed to parse in earlier sets (the English part was 3,519 rows and mostly negatives), and "no tool" was reduced from 28 to 6 percent because the benchmark never asks for it. No closed model produced a label. A disjointness gate aborts the build if any training tool name or query appears in FunctionChat-Bench.

Two later additions (sets v4 and v5, same day): 7,745 Korean rewrites of glaive requests produced by our own Palette-K-Midm with the schemas and gold calls left untouched (Korean queries over thousands of tools instead of 57), no-parameter tools upweighted, and every no-parameter schema rendered in one of the three common JSON shapes at random. The last one came from reading failures: 45 of the benchmark's 57 no-parameter tools use `{"type":"object","properties":{},"required":[]}`, our set had almost only bare `{}`, and the model had learned the shape instead of the rule. Each set was scored before the next was built; Table 2 carries the published one and the text names the others.

Post-training recipe: full-parameter SFT, one epoch then a second at a lower rate, followed by a similarity-reward RL stage (group-relative policy optimisation against the gold call, reward = format term plus argument similarity as in STAR, Ni et al. 2026, with our own labels as the only signal). Each stage is scored on the benchmark and a stage that lowers the score is not published; Table 2 reports the published one. For the 12-layer model the RL stage tied the SFT score (11.4 percent before and after), and the published weights carry it.

What the RL stage measured. Three runs on the 12-layer model, 16 prompts x 8 samples per step, KL to the frozen SFT model. Run 1 drew prompts from the post-training set: the group mean reward was 0.97-0.99 over the first 100 steps, the eight samples of a group almost always agreed, and the benchmark did not move. Run 2 drew the same queries with every tool renamed to a name that never appears in training: reward 0.84-0.99 over 140 steps, because the model reproduces the gold call for a query it was trained on even when the tool is called something else. Run 3 used prompts the model had never seen (1,575 Korean rewrites of glaive requests held out of post-training, their renamed copies, and 3,000 held-out English rows): reward 0.70-0.85 with exact matches at 0.58-0.69, and real disagreement inside each group, the first run with a gradient worth following. After 200 of those steps the benchmark was still 11.4 percent, identical in every condition. A model this small memorises its post-training queries, so on-policy RL can only teach what SFT has not already fixed, and 200 steps of it on 3,200 novel prompts did not reach the benchmark's tools. The RL stage is kept in the recipe as a measured negative, not as a source of the published number.

5. Evaluation

FunctionChat-Bench SingleCall (Kakao, 2024, Apache 2.0): 25 functions, 4 Korean queries each, 5 tool conditions (exact, 4 random, 4 close, 8 random, 8 close) = 500 items. The official protocol uses GPT-4 as judge; we score with exact match on function name and arguments (numbers

compared as numbers, acceptable alternatives honoured) so the number is reproducible offline. Every model receives the same tools and query, rendered in its own documented format.

Table 2. Call accuracy (exact match) on SingleCall, by tool condition, percent. Songgot rows are written by the build from the run logs; a row reads "training" until its run has finished.

MODEL	PARAMS	EXACT	4_RANDOM	4_CLOSE	8_RANDOM	8_C
Songgot-nano (Mac, 320M tokens)	39M	0.0	0.0	0.0	0.0	0.0
Songgot (8xH100, 6B tokens)	50M	26.0	12.0	10.0	7.0	2.0
Songgot-M (16 layers, 8xH100, 6B tokens)	126M	32.0	13.0	8.0	7.0	0.0
Needle 2	45M	0.0	0.0	0.0	0.0	0.0
FunctionGemma-270M	270M	3.0	5.0	1.0	1.0	1.0
Qwen3-0.6B	600M	48.0	49.0	45.0	37.0	37.(
Qwen3.5-0.8B	800M	51.0	48.0	41.0	52.0	34.(

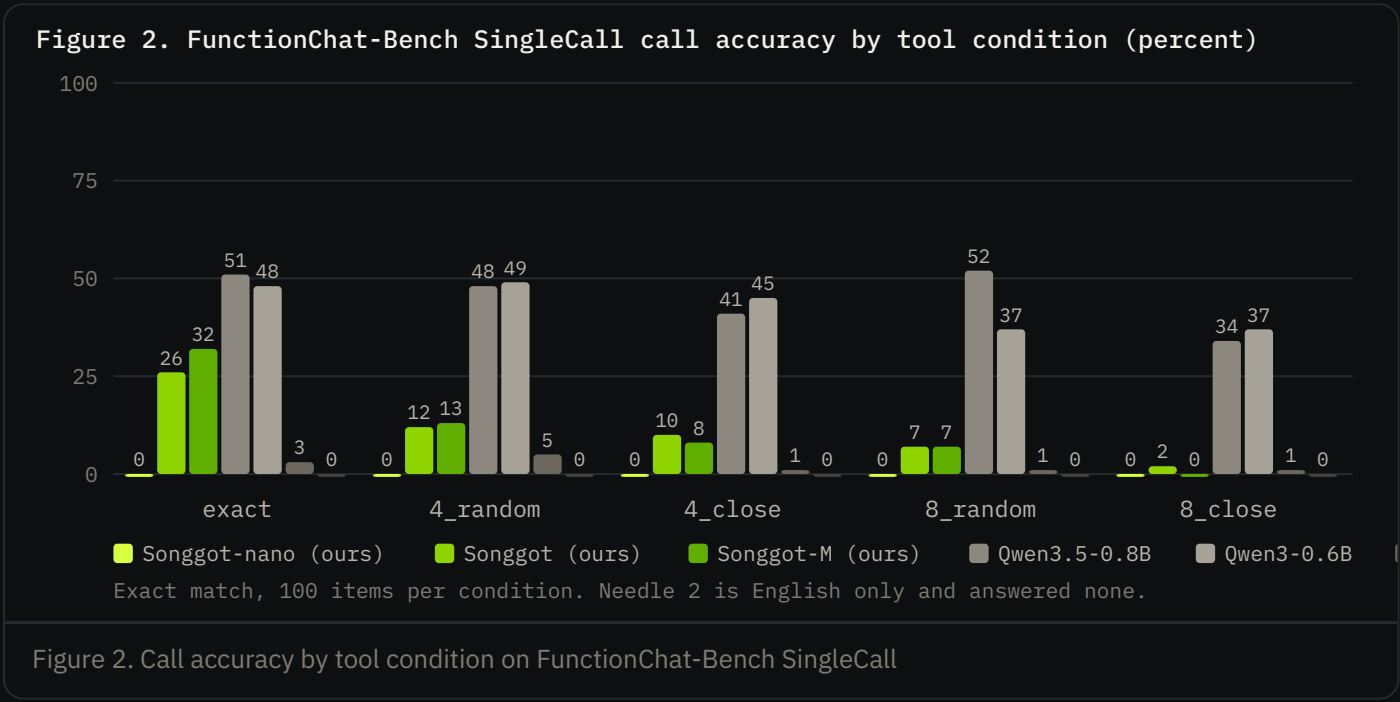

Figure 2. Call accuracy by tool condition on FunctionChat-Bench SingleCall

6. Honest reading

- Songgot-nano, 320M pretraining tokens and one post-training epoch on the v3 set, scores 0.0 on both call and name accuracy. It answers with tool names from its own post-training catalogue (create_event, get_subway_arrival, lookup_exchange_rate) instead of the names in the prompt, and "none" for 105 of the 500 items. The 12-layer model pretrained on 6B tokens copies the right name 54 percent of the time from the same data. At 320M tokens the model has not learned to copy from context at all; the Needle 2 recipe used 200B. The

nano weights stay published as the measured floor of the recipe, and the 12-layer model is the one in the app.

- Songgot-M, 16 layers and hidden 768 (126M parameters), pretrained on the same 6B tokens and post-trained on the same v5 set, scores 12.0 percent against the 12-layer model's 11.4: 32 against 26 percent with a single tool, and the same or worse with four and eight tools (13/8/7/0 against 12/10/7/2). Name accuracy is 53.0 against 53.6. Two and a half times the parameters buy three more correct calls out of 500, all of them in the easiest condition. At this token count the bottleneck is what the model has read, not how many parameters it has, and the next experiment is more pretraining tokens on the 12-layer size, not a larger model.
- The template-generated Korean data is narrow by construction; a teacher-generated set from our own Palette-K-Midm was planned and is queued behind GPU availability.
- The preview checkpoint used to open the demo on 2026-09-10 is an early one (65M tokens, 24,000 post-training examples) and is labelled as such in the table.

7. Release

Weights and tokenizer: hf.co/palette-lab/songgot. Code, data generators, scorer, paper: github.com/hanishkeloth/songgot. Licence Apache 2.0. Korean Wikipedia attribution and CC BY-SA notice in the model card.

8. On the device

Songgot Pocket (hanishkeloth.github.io/songgot/app) is the 12-layer model running inside the browser: llama.cpp compiled to WebAssembly (wllama 3.6.1), the Q8_0 GGUF (54 MB) fetched once and kept in the browser cache, no server in the loop. It installs as a web app on iOS, Android and desktop and keeps working with the network off. The app shows the model at most six candidate tools per request (chosen by character bigram overlap with a 58-tool Korean catalogue), renders the returned call as a card, and performs the calls it can do locally (alarms, timers, notes, calendar files) while handing the rest to the right site when online. Measured on 2026-09-10 in Chrome on an Apple M5 Max, single thread: 1.5 s per request from prompt to parsed call with the 12-layer weights (755 prompt tokens, 31 generated, 21 tokens per second), 1.7 s with the 39M weights.

Questions and answers

▶ **What is Songgot?**

▶ **How big is it?**

▶ **How is it different from Needle 2?**

▶ **What data was used?**

▶ **How is it evaluated?**

▶ **Where are the weights and code?**

About the author

Songgot is built by **Hanish Keloth**, CTO at Palette (Seoul and Bengaluru), where he leads Palette OS, a company operating system in which governed AI agents draft the documents Korean companies run on. Palette publishes its Korean models and benchmarks in the open: Palette-K-Midm (first of seven on PALETTE-BENCH-KO v0.2, 2026-08-21), Palette-K-Doc, Palette-K-Speech and Palette Video, all at hf.co/palette-lab. Songgot continues that line at the smallest possible size.

GitHub · LinkedIn · Hugging Face