

Udgam-384M: a small on-device tool-calling model trained from scratch

UDGAM LABS VERSION 1.0.0

Abstract

Udgam-384M is a family of three English language models with 384.4M parameters each. Udgam Labs trained them from scratch on roughly 8B tokens and released them with open weights under Apache-2.0 on 10 October 2026:

- **Udgam-384M-Base** has a 2,048-token context.
- **Udgam-384M-Base-4K** adds a short pretraining stage at 4,096 tokens.
- **Udgam-384M-Chat** is post-trained from Base-4K to be the language layer of an app on the device. Given the app's tools, it turns a request into a function call, fills a JSON schema from a document, proposes a shell command for a person to review, or asks or declines.

The models use the Gemma 4 architecture and chat format [1], so the same weights run unchanged in transformers, llama.cpp, Ollama, LM Studio and MLX. The 4-bit GGUF file is 238 MB.

We compared Udgam-384M-Chat with Granite 4.0 350M, Granite 4.0 H 350M, SmoLLM2-360M-Instruct and Qwen3-0.6B in one harness (greedy decoding, Q8_0, thinking off).

- **Held-out tests.** These use tools, document types and shell tasks never seen in training, but they follow this model's conventions and so favour it. On them it makes the right relevance decision for 75% of requests (others 18.3–66.7%), returns schema-valid JSON for 56% of unseen schemas (0–38%) and chooses the right shell behaviour for 85.9% of tasks (39.1–67.2%).
- **Exact calls.** Only 31.4% of its held-out calls are exactly right, against 45.7% for both Granite models.
- **Neutral benchmarks.** It trails the Granite models and Qwen3-0.6B: BFCL v3 simple 60.0% (Granite 4.0 350M: 90.7%), IFEval prompt-level strict 26.1% (47.0–58.0%). On BFCL irrelevance it scores 68.0%, against 88.0% for Qwen3-0.6B.
- **Safety.** On XSTest it declines 54.0% of unsafe prompts and over-refuses 14.8% of safe ones. It is released as not safety-hardened.
- **Builds and speed.** The Q8_0 and Q6_K builds are within noise of f16. On an Apple M6 Mac mini, the Q4_K_M file generated 444 tokens per second on the GPU at release, and a complete tool call takes about 0.09 seconds on the GPU and 0.25 seconds on 4 CPU threads.

A re-run of the main evaluation on the released Q8_0 file on 17 October 2026 reproduced every published number to within 0.3 points.

1. Introduction

Much of what an app needs from a language model is narrow: given the functions the app offers, turn the user's request into the right call, or recognise that none of them fits. Doing this on the device keeps requests private, works offline and costs nothing per call. It also limits the model to a few hundred million parameters.

Udgam-384M was built for that step, under four constraints:

1. **From scratch and open:** random initialisation, our own tokenizer, licensed data, and Apache-2.0 weights.
2. **Small:** under 400M parameters, with a 4-bit file under 250 MB.
3. **Standard formats:** an architecture and chat format that existing runtimes already support.
4. **The app stays in charge:** the model proposes, and the app checks each call, runs it and confirms anything with consequences.

The chat model has four jobs:

1. **App actions.** "Wake me at 6:45, label it gym" becomes `set_alarm{time:"06:45", label:"gym"}`. Relative dates are resolved from the date-time in the system message.
2. **Document → JSON.** Fill a supplied JSON schema from a receipt, email, form or OCR text.
3. **Natural language → shell.** Propose one command for a person to review, as a `run_shell_command` call with a `requires_confirmation` flag or as a plain command block.
4. **Relevance.** Decline when no offered tool fits, ask when a required detail is missing, and refuse requests that should not be carried out.

We use two kinds of test, and each comparison says which:

- **Held-out tests** are our own sets for the four jobs. They are built on tools, document types and shell tasks that were kept out of all training data, but they use this model's prompt conventions.
- **Neutral tests** are public benchmarks: BFCL v3 [2, 3], IFEval [4] and XSTest [5].

2. Model overview

	UDGAM - 384M - BASE	UDGAM - 384M - BASE - 4K	UDGAM - 384M - CHAT
What it is	Pretrained base model; continues text	The base after a short pretraining stage at 4,096 tokens	Post-trained from Base-4K for chat and tool use
Context	2,048 tokens	4,096 tokens	4,096 tokens
Safety training	none	none	refusal examples; not safety-hardened
Hugging Face	udgamlabs/Udgam-384M-Base	udgamlabs/Udgam-384M-Base-4K	udgamlabs/Udgam-384M-Chat

All three have 384.4M parameters and an Apache-2.0 licence. Each repository holds:

- the bf16 weights (`model.safetensors` , 769 MB);
- the configuration and tokenizer files;
- the chat template.

Each base repository also has f16 and Q8_0 GGUF files. The chat model has seven more builds:

BUILD	OF	UDGAM - 384M - CHAT	REPOSITORY	SIZE	USE
GGUF f16			Udgam-384M-Chat-GGUF	770 MB	reference
GGUF Q8_0			same	410 MB	default
GGUF Q6_K			same	317 MB	
GGUF Q4_K_M, importance matrix			same	238 MB	phones and low-memory devices
GGUF Q4_0, importance matrix			same	227 MB	runtimes without k-quants
MLX fp16			Udgam-384M-Chat-MLX	769 MB	Apple silicon
MLX 4-bit, group size 64			Udgam-384M-Chat-MLX-4bit	216 MB	Apple silicon

The GGUF repository also holds the importance matrix and two Ollama Modelfiles. The GitHub repository [udgamlabs/udgam-384m](#) holds the documentation, six runnable examples, SHA-256 checksums and a [walkthrough page](#).

These replies were recorded on 10 October 2026 with the GitHub examples (Ollama, Q8_0, temperature 0.1, thinking off, with alarm, message and weather tools offered). They show typical behaviour, not a representative sample:

REQUEST	WHAT THE MODEL RETURNED
"Wake me up at 6:45 tomorrow, label it gym."	<code>set_alarm({"label": "Gym", "time": "06:45"})</code>
"Text Priya that I'll be 10 minutes late."	<code>send_message({"contact": "Priya", "text": "I'll be 10 minutes late"})</code>
"Send a message to Rahul."	"What should the message to Rahul say?"
"Book me a table for two at 8pm."	"I can't book restaurant tables here. Want me to set an alarm for 8 pm so you can ring them?"

3. Architecture

Udgam-384M is a decoder-only transformer. Its `config.json` declares the `Gemma4ForCausalLM` class of `transformers` (model type `gemma4_text`) [1], so `transformers`, `llama.cpp` [6], `Ollama` [7] and `MLX` [8] load it with their existing Gemma 4 code. The details below come from the released `config.json` and weights.

UDGAM-384M (ALL THREE MODELS)

Layers	20
Hidden width	1,024
Attention heads	16 query and 4 key-value heads (grouped-query attention [9])
Head size	64 in sliding-window layers, 128 in global layers
Local and global attention	Three sliding-window layers (window 512 tokens), then one global layer, repeated [10, 11]; layers 4, 8, 12, 16 and 20 are global
Position encoding	Rotary [12]: base 10,000 in sliding-window layers; base 1,000,000, proportional, in global layers
MLP	Gated, GELU (tanh approximation) [13], width 4,096
Normalisation	RMSNorm [14] before and after the attention block and the MLP; RMS normalisation of queries and keys [15]; a learned scalar per layer
Output	Final-logit soft-cap of 15 [11]
Vocabulary	32,768 tokens
Embeddings	Untied input table and output projection, each $32,768 \times 1,024$ (33.55M parameters)
Not used	Per-layer embeddings, mixture of experts, shared key-value caches
Parameters	384,389,268 (384.4M), of which 317.3M are in the 20 layers
Context	2,048 positions (Base); 4,096 (Base-4K, Chat), in <code>config.json</code> and in the GGUF files
Weights	bf16

Why this layout. All the major runtimes support the Gemma 4 text architecture natively. It also keeps features that suit a small on-device model:

- sliding-window attention, which keeps attention cost and cache memory small on long prompts;
- a logit soft-cap;
- query-key normalisation.

Before training, we checked that a small, randomly initialised model in this layout gives the same outputs in our training code, transformers, llama.cpp and MLX. For the released chat model:

- transformers (fp32) and llama.cpp (f16) give identical greedy outputs on 8 of 8 test prompts;
- MLX fp16 matches on 11 of 12 (the twelfth diverges late, at a near-tie);

- MLX 4-bit matches on 4 of 12, as is usual for 4-bit weights;
- Ollama's built-in Gemma 4 renderer produces the same tokens as the template on 11 of 11 cases.

For Base-4K, llama.cpp matches transformers token for token on prompts of 3,299 and 3,900 tokens.

Tokenizer. A byte-level BPE tokenizer with 32,768 tokens, trained by us.

- It includes Gemma 4's special tokens for turns, tool declarations, tool calls, tool responses and thoughts, so the official Gemma 4 chat template works unchanged. The template in the repositories is identical to the official one.
- llama.cpp recognises its pre-tokenizer, so GGUF conversion needs no patches.

4. Training (high level)

All three models were trained by Udgam Labs from random initialisation; no weights from any other model were used.

1. **Pretraining → Udgam-384M-Base.** Roughly 8B tokens of licensed public and synthetic English text, at a sequence length of 2,048. The text included structured and conversational material (tool calls, JSON and chat), so the base model already knows the Gemma 4 tool-call format (section 6.6).
2. **Context extension → Udgam-384M-Base-4K.** A short stage of continued pretraining at 4,096 tokens.
3. **Supervised fine-tuning → Udgam-384M-Chat.** From Base-4K, on the four jobs, general conversation, declining harmful requests, and helping with benign requests that only sound alarming.

Test-set hygiene.

- The tools, document types and shell tasks of the held-out tests, and a range of dates, were kept out of all training data, pretraining included.
- Training text was checked for exact matches and shared 13-word sequences against the public benchmarks in this report and our held-out sets, and matching documents were dropped.
- The safety training prompts were also checked against XSTest, the Aegis 2.0 test split [17] and OR-Bench-Hard [18]. These three are used only for evaluation.

This report does not describe data proportions, training settings or compute.

5. Inference

5.1 Chat and tool-call format

The models use the standard Gemma 4 chat format. A tool turn looks like this:

```
<bos><|turn>system
Current date-time: Tuesday, 2027-08-17 09:12 (Europe/London, UTC+01:00)
Device: Android phone<|tool>declaration:set_alarm{description:<|>Set an alarm
at a local time.<|>,parameters:{properties:{label:{description:<|>Optional
label<|>,type:<|>STRING<|>},time:{description:<|>HH:MM 24h<|>,type:
<|>STRING<|>}},required:[<|>time<|>],type:<|>OBJECT<|>}}<tool>
<turn|>
<|turn>user
Wake me up at 6:45 tomorrow, label it gym.<turn|>
<|turn>model
<|tool_call>call:set_alarm{label:<|>gym<|>,time:<|>06:45<|>}}
<tool_call|><|tool_response>
```

The model stops at `<|tool_response>`. The app runs the tool, sends the result back as a `tool` message, and the model writes the final reply. Ollama and `llama-server --jinja` return the call as structured `tool_calls`.

Thinking is optional. With thinking on, `<|think|>` follows `<|turn>system`, and tool turns start with a one-line thought. It gives no consistent gain for this model (section 7.5), so we recommend turning it off.

Conventions. The model works best with the conventions of its training data:

- the date-time, time zone and device in the system message;
- one JSON schema per extraction;
- only the tools the user may use right now;
- ISO dates in arguments.

5.2 Running the chat model

- **Ollama (recommended):** the Modelfile in the GGUF repository uses Ollama's built-in Gemma 4 renderer and parser and the settings of section 5.4. Ollama turns thinking on by default for Gemma 4 models; send `"think": false`.
- **llama.cpp:** `llama-server -m Udgam-384M-Chat-Q8_0.gguf -c 4096 --jinja --reasoning off` serves an OpenAI-compatible API with parsed tool calls. Without `--reasoning off`, llama-server renders the template with thinking on and adds `<|think|>` and a newline to the system turn. For tool use, pass `--reasoning off`, or `"chat_template_kwargs": {"enable_thinking": false}` per request.

- **transformers** (tested with 5.18): `tokenizer.apply_chat_template(messages, tools=tools, add_generation_prompt=True, return_dict=True, return_tensors="pt")` , then `model.generate` . For bit-exact greedy results use fp32 or llama.cpp; bf16 generation depends on the batch shape.
- **MLX** (mlx-lm 0.32): `mlx_lm.generate` or `mlx_lm.server` with either MLX repository.
- **LM Studio**: the GGUF files carry the chat template. Set the context to 4,096, and for the MLX builds set Max Concurrent Predictions to 1.

On 17 October 2026 we ran the `set_alarm` example above with the released Q8_0 file in `llama-server --jinja --reasoning off` (llama.cpp build d7a695e). It returned the parsed call `set_alarm` with `{"label": "Gym", "time": "06:45"}`.

5.3 Exactly one <bos>

Every training document started with `<bos>` , so every input should start with exactly one. The chat template begins with `<bos>` , so chat use is safe in every runtime. Plain-text use of the base models needs care. We checked each path on 17 October 2026:

HOW THE INPUT IS BUILT	<BOS> TOKENS AT THE START
<code>transformers, apply_chat_template(..., tokenize=True)</code>	one
<code>transformers, tokenizer(text)</code>	one with the current tokenizer files; none with the files first released
<code>transformers, tokenizer("<bos>" + text)</code>	two with the current files; one with the first release
<code>transformers, tokenizer("<bos>" + text, add_special_tokens=False)</code>	one with either version (recommended)
<code>llama.cpp, plain prompt (llama-cli -p , /completion)</code>	one (the GGUF files set <code>add_bos_token</code>)
<code>llama-server --jinja , /v1/chat/completions</code>	one
Ollama, <code>/api/chat</code> and raw <code>/api/generate</code>	one
<code>mlx-lm 0.32, --ignore-chat-template</code>	the tokenizer default: one with the current files, so do not type a literal <code><bos></code>

Tokenizer versions. Since 12 October 2026, plain text gets one `<bos>` from the tokenizer, as in training and in the GGUF builds. The `tokenizer.json` files first published on 10 October added none; the weights, the GGUF files and chat-template use are the same in both versions.

This form works with both versions:

```

from transformers import AutoTokenizer, AutoModelForCausalLM
tok = AutoTokenizer.from_pretrained("udgamlabs/Udgam-384M-Base")
model = AutoModelForCausalLM.from_pretrained("udgamlabs/Udgam-384M-Base")
ids = tok("<bos>The capital of France is", add_special_tokens=False,
return_tensors="pt").input_ids
print(tok.decode(model.generate(ids, max_new_tokens=30, do_sample=False)[0]))

```

Section 7.7 measures what a missing or doubled `<bos>` costs. No number in this report is affected: every chat-model evaluation used the chat template, and every base-model measurement put one `<bos>` before each document.

5.4 Recommended settings

SETTING	TOOL AND APP USE (DEFAULT MODELFILE)	OPEN CHAT (MODELFILE.CHAT)
Temperature	0.1	0.5
top-p / top-k	0.9 / 40	0.9 / 40
Repeat penalty (last 64 tokens)	1.1	1.1
Maximum new tokens / context	1,024 / 4,096	1,024 / 4,096
Thinking	off	off

Use greedy decoding when tool calls must be reproducible. In transformers, `generation_config.json` sets temperature 0.1, top-p 0.9 and top-k 40 but no repetition penalty, because transformers would also apply it to the prompt and penalise tool names. Section 7.4 explains the low temperature.

5.5 Speed and memory

Release-time measurement (9 October 2026). All five models were measured the same way on an Apple M6 Mac mini (12 CPU cores, 24 GB of unified memory):

- llama.cpp build d7a695e, `llama-bench` , 512-token prompt, 128 generated tokens, 3 repetitions;
- the GPU (Metal) and 4 CPU threads;
- each model's Q4_K_M file. We converted and quantised the other models without an importance matrix.

Peak memory is the macOS peak memory footprint of the CPU run. On the GPU, the weights sit in Metal buffers, which that figure does not fully count.

MODEL (Q4_K_M)	PARAMETERS (LLAMA.CPP COUNT)	GPU PROMPT / GENERATE, TOKENS/S	4 CPU THREADS PROMPT / GENERATE, TOKENS/S	PEAK MEMORY, CPU RUN
Udgam-384M-Chat	384M	14,191 / 444	983 / 365	323 MB
Udgam-384M-Chat (Q8_0)	384M	14,561 / 312	1,271 / 297	480 MB
Granite 4.0 350M	352M	17,626 / 411	1,187 / 348	349 MB
Granite 4.0 H 350M	340M	5,175 / 262	966 / 307	352 MB
SmolLM2-360M-Instruct	362M	14,175 / 358	1,378 / 253	238 MB
Qwen3-0.6B	752M	9,774 / 282	657 / 213	579 MB

Udgam-384M-Chat generated fastest of the five on both the GPU and the CPU. llama.cpp counts Qwen3-0.6B's tied embedding table twice; its developers report 0.6B parameters [19].

Re-measurement on the released files (17 October 2026). We used the same build and settings with 5 repetitions, in 4 rounds about 2.5 minutes apart. Other jobs were using the Mac's CPU and GPU heavily (load average 5 to 19 on 12 cores). The table gives the median of the four rounds, with the fastest round in brackets:

RELEASED FILE	GPU PROMPT / GENERATE, TOKENS/S	4 CPU THREADS PROMPT / GENERATE, TOKENS/S	PEAK MEMORY, CPU RUN
Q4_K_M	12,196 (13,043) / 317 (325)	702 (830) / 242 (320)	324–328 MB
Q8_0	12,938 (13,239) / 228 (268)	914 (1,022) / 208 (244)	479–482 MB

- **Memory** reproduces the release-time figures.
- **Speed** was lower in every round because the GPU and CPU were shared. Even the fastest GPU round for Q4_K_M was 27% below release, and one GPU round fell to 62 tokens/s.
- **Which numbers to use.** We have no record of the load at release time, so the release-time speeds are our best estimate for this Mac, and the 17 October speeds are lower bounds.

One tool call, end to end. On 17 October we also timed the `set_alarm` request of section 5.1 in `llama-server --jinja --reasoning off`: greedy, one request at a time, prompt cache off, 2 warm-up requests and then 20 timed ones. Each request sends 140 prompt tokens and receives 26 tokens, ending in the parsed call.

RELEASED FILE	DEVICE	MEDIAN SECONDS PER CALL	90TH PERCENTILE	PEAK MEMORY OF THE SERVER
Q4_K_M	GPU (Metal)	0.089	0.102	157 MB, plus Metal buffers
Q4_K_M	4 CPU threads	0.252	0.281	347 MB
Q8_0	GPU (Metal)	0.126	0.140	152 MB, plus Metal buffers
Q8_0	4 CPU threads	0.265	0.278	505 MB

All 80 timed requests returned the same call. The times include the HTTP round trip, template rendering and tool-call parsing, but not model loading. The load average during these runs was about 5. We have not measured Udgam-384M on a phone, which will be slower.

6. Evaluation

6.1 Setup

All numbers were measured by Udgam Labs unless labelled as reported by others.

Held-out tests (274 items): built on tools, document types and shell tasks excluded from all training data, and on a held-out range of dates.

SET	ITEMS	WHAT IS EXPECTED
App actions (20 unseen tools)	100	70 calls (10 of them parallel), 20 questions about a missing detail, 10 no call
Relevance (an unseen tool in every request)	60	10 calls, 6 questions, 22 answers without a tool, 14 declines because no tool fits, 8 refusals of harmful requests
Document → JSON (10 unseen document types)	50	a schema-valid extraction
Shell (16 unseen tasks)	64	52 commands (4 flagged for confirmation), 6 requests for confirmation, 4 refusals with a safer alternative, 2 questions; 26 commands are run in a sandbox and their output compared with a reference

Neutral tests:

- **BFCL v3** [2, 3]: fixed random subsets of simple (150 of 400), multiple (100 of 200), parallel (50 of 200) and irrelevance (150 of 240), with no system prompt. Scored by our re-implementation of BFCL's syntax-tree checker. The scores are not comparable with the official leaderboard.
- **IFEval** [4]: all 541 prompts, scored by the IFEval code of lm-evaluation-harness 0.4.9 [20].

- **XSTest** [5]: 200 unsafe and 250 safe prompts, up to 256 new tokens. A reply counts as declined when its first 300 characters contain a refusal phrase from a fixed list. The same classifier is used for every model.

Settings.

- Every model ran in llama.cpp (build d7a695e) on the Mac of section 5.5, at Q8_0, with greedy decoding and thinking off.
- Each model used its own chat template, tool-call format and end-of-turn tokens, with the same limits on new tokens: 640, 900 for documents, 400 for BFCL and 768 for IFEval.
- Tool calls were converted to one canonical form and scored by the same code.
- The comparison ran on 9 October 2026, the quantisation study on 8 and 9 October.

Uncertainty. Near 50%, the 95% binomial interval is about ± 12 points for 60 to 70 items, ± 8 for 150 and ± 4 for 541.

6.2 Re-run on the released file

On 17 October 2026 we re-ran the held-out, BFCL and IFEval items (1,265 generations) on the released `Udgam-384M-Chat-Q8_0.gguhf`, with the original harness, prompts, scorer and llama.cpp build. The shell execution checks need a Linux sandbox and were not re-run.

TEST	PUBLISHED	RE - RUN
Held-out app actions: exact call / asks when a detail is missing	31.4 / 20	31.4 / 20
Held-out relevance: right decision	75	75
Held-out document → JSON: schema-valid / field F1	56 / 68.7	56 / 69.0
Held-out shell: right behaviour	85.9	85.9
BFCL v3 simple / multiple / parallel / irrelevance	60 / 67 / 46 / 68	60 / 67 / 46 / 68
IFEval prompt-level / instruction-level strict	26.1 / 40.4	26.1 / 40.3

1,258 of the 1,265 outputs were identical to the published run. The other seven differ because llama-server batches eight requests, which changes the order of floating-point operations.

6.3 Comparison with other small open models

TEST (Q8_0, GREEDY, THINKING OFF)	UDGAM-384M-CHAT	UDGAM-384M-CHAT, Q4_K_M	GRANITE 4.0 350M	GRANITE 4.0 H 350M	SMOLLM2-360M-INSTRUCT	QWEN3-0.6B
Held-out app actions: exact call	31.4	28.6	45.7	45.7	1.4	44.3
Held-out app actions: right call / no-call decision	74	74	70	72	66	70
Held-out app actions: asks when a detail is missing	20	25	0	5	0	20
Held-out relevance: right decision	75	68.3	18.3	20	46.7	66.7
Held-out relevance: declines when no tool fits	74	66	2	4	44	66
Held-out document → JSON: schema-valid	56	60	32	38	0	24
Held-out document → JSON: field F1	68.7	69.7	52.7	45	9.3	63.7
Held-out shell: right behaviour	85.9	85.9	59.4	67.2	39.1	67.2
Held-out shell: valid bash syntax	96.6	96.6	97.7	98	100	95.1
Held-out shell: correct output when run (26)	7.7	7.7	3.8	7.7	0	0
BFCL v3 simple (150)	60	62.7	90.7	88	4.7	80
BFCL v3 multiple (100)	67	64	88	90	5	76
BFCL v3 parallel (50)	46	50	82	76	0	64
BFCL v3 irrelevance (150)	68	65.3	55.3	66.7	75.3	88
IFEval prompt-level strict (541)	26.1	25.5	47	53.4	31.4	58
IFEval instruction-level strict	40.4	39.7	58.9	65.6	42.4	68.7

TEST (Q8_0, GREEDY, THINKING OFF)	UDGAM-384M-CHAT	UDGAM-384M-CHAT, Q4_K_M	GRANITE 4.0 350M	GRANITE 4.0 H 350M	SMOLLM2-360M-INSTRUCT	QWEN3-0.6B
XSTest unsafe prompts declined (higher is safer)	54.0	54.0	80.0	92.5	27.5	17.0
XSTest safe prompts over-refused (lower is better)	14.8	14.8	37.2	44.8	10.0	3.2

Bold marks the best value in each row. The Q4_K_M column is Udgam's phone build.

Where Udgam-384M-Chat is ahead: deciding whether to act.

- On our relevance set it declines 74% of requests that no offered tool fits; both Granite models call a tool almost every time.
- It asks for a missing detail as often as Qwen3-0.6B and more often than the others.
- It is the most likely to return schema-valid JSON and to choose the right kind of shell answer.

Where it is behind: exact calls and instruction following. Both Granite models are far ahead on BFCL (simple 88–91% against 60%, parallel 76–82% against 46%), on held-out exact calls (45.7% against 31.4%) and on IFEval (47–53% against 26%). Qwen3-0.6B is also ahead on BFCL and IFEval.

Fairness notes:

- **Conventions.** The held-out sets use this model's conventions (the date-time and device header, ISO dates, our rules for free-text arguments), so other models lose some points to convention rather than ability.
- **Neutral relevance.** On BFCL irrelevance, Qwen3-0.6B (88%) and SmoLLM2 (75%) decline more often than Udgam (68%). Udgam's relevance lead shows mainly on our own set (section 7.2).
- **SmoLLM2's tool prompt.** SmoLLM2's chat template has no tool support, so it was given the function-calling prompt from its model card, which was written for the 1.7B model. It mostly copied that prompt's example call, so its near-zero tool scores mean "not a tool model".
- **Thinking.** Qwen3 ran with thinking off; Granite and SmoLLM2 have no thinking mode.
- **<bos> .** No other model's GGUF file or chat template adds a **<bos>** , so tokenising their prompts without special tokens matched their native behaviour.
- **XSTest scoring.** On Udgam-384M-Chat, the XSTest refusal-phrase classifier agreed with an LLM judge (gpt-oss-20b [16]) within 3 points (section 8.2).

- **Not compared.** Gemma 3 270M and FunctionGemma-270M, whose weights were gated behind Google's Gemma terms at the time.

6.4 Numbers reported by the other models' developers

These figures come from the developers' own pipelines; we did not reproduce them.

- **Granite 4.0 350M and H 350M [21]:**
- IBM reports IFEval prompt-level strict scores of 49.17 and 55.64. We measured 47.0 and 53.4, within 2.3 points.
- IBM reports BFCL v3 scores of 39.32 and 43.32. These are overall scores across BFCL's own categories, not comparable with our subsets.
- IBM reports pretraining Granite 4.0 350M on about 15T tokens.
- **SmolLM2-360M [22]:** pretrained on 4T tokens.
- **Qwen3-0.6B [19]:** 0.6B parameters, 0.44B outside the embeddings.

Udgam-384M was pretrained on roughly 8B tokens: about 500 times fewer than SmolLM2-360M and about 1,900 times fewer than Granite 4.0 350M.

6.5 Quantisation

Every build of Udgam-384M-Chat went through the same evaluation, with thinking off:

TEST (THINKING OFF)	F16	Q8_0	Q6_K	Q4_K_M, IMPORTANCE MATRIX	Q4_K_M, NONE	Q4_0, IMPORTANCE MATRIX
Held-out app actions: exact call	32.9	31.4	32.9	28.6	30	28.6
Held-out app actions: asks when a detail is missing	20	20	30	25	30	25
Held-out relevance: right decision	76.7	75	76.7	68.3	78.3	71.7
Held-out relevance: declines when no tool fits	76	74	76	66	78	70
Held-out document → JSON: schema-valid	58	56	58	60	60	44
Held-out document → JSON: field F1	68.8	68.7	70.3	69.7	70	69.7
Held-out shell: right behaviour	85.9	85.9	87.5	85.9	85.9	84.4
BFCL v3 simple	60	60	60	62.7	60	58
BFCL v3 multiple	67	67	65	64	67	57
BFCL v3 parallel	50	46	48	50	48	44
BFCL v3 irrelevance	70	68	70.7	65.3	66	68.7
IFEval prompt-level strict	26.2	26.1	27.2	25.5	24.6	27.4
Perplexity, held-out chat and tool-use text	3.193	3.195	3.209	3.326	3.375	3.458

The same suite was also run with thinking on, giving 16 numbers per mode. Averaged over all 32, the change against f16 is:

BUILD	MEAN CHANGE	MEAN ABSOLUTE CHANGE	WORST SINGLE CHANGE
Q8_0	-0.8 points	1.1	
Q6_K	+0.1 points	1.4	
Q4_K_M, importance matrix	-0.5 points	2.5	- 10, on declines when no tool fits
Q4_0, importance matrix	-0.9 points	4.5	- 14

- **Perplexity** rises by 0.07% for Q8_0, 0.5% for Q6_K and 4.2% for Q4_K_M. Without the importance matrix, Q4_K_M rises by 5.7%.

- Q8_0 is the default: it is lossless within noise.
- Q4_K_M with the importance matrix is the build for phones. Its clearest loss is in declines when no tool fits: 76% falls to 66%.
- Q4_0 is only for runtimes without k-quants.
- The importance matrix was computed from our own fine-tuning data.

6.6 Base models

Held-out text. Bits per byte at 2,048 tokens, from our training-time harness. Both models saw the same fixed samples of three validation sets, with one `<bos>` per document.

BITS PER BYTE (LOWER IS BETTER)	UDGAM-384M-BASE	UDGAM-384M-BASE-4K
Mixed text	0.6622	0.6632 (+0.15%)
Web text	0.8552	0.8548
Task text (tool calls, JSON, chat)	0.3910	0.3920

On the full validation sets used during pretraining, Udgam-384M-Base scores 0.668, 0.865 and 0.408.

Long context. We measured three things: bits per byte on 400 held-out long documents, and two synthetic tests of 100 trials each.

TEST	UDGAM-384M-BASE	UDGAM-384M-BASE-4K
Bits per byte at positions 2,048–3,072 / 3,072–4,096	0.769 / 0.848	0.734 / 0.730
Recall of a 5-digit code hidden in filler text, at 1K / 2K / 3.5K / 7K tokens	100 / 100 / 1 / 0%	100 / 100 / 83 / 1%
Copying a 48-token span, next-token accuracy at 1K / 3.5K / 7K tokens	0.98 / 0.18 / 0.14	0.98 / 0.86 / 0.22

The 4K stage leaves short-context quality unchanged and makes positions up to 4,096 usable. Neither model works beyond 4,096 tokens.

Tool calls from the base model. Prompted with the chat template in the harness of section 6.1 (f16), Base-4K already makes tool calls. It scores BFCL v3 simple 66.0%, multiple 75.0%, parallel 48.0% and irrelevance 11.3%, and 24.3% exact calls on the held-out app actions.

7. Analysis

7.1 Deciding is easier than doing

On the 70 held-out app actions that need a call, Udgam-384M-Chat:

- makes a call 65 times;
- uses the right tool or tools 41 times (58.6%);
- gets every argument right 22 times (31.4%).

Both Granite models pick the right tool for 84–86% of these requests, so most of their lead comes from choosing the tool. On unseen tools, Udgam often picks a plausible but wrong one.

The decision itself is stronger: 74% of call-or-no-call decisions are right, and 37 of 50 relevance requests that need no call are declined. Asking is the weak spot. Of the 20 app actions that lack a required detail, it asks in 4 and guesses a value and calls the tool in 15. The other models do no better: 0–1 of 20 for Granite and SmolLM2, 4 for Qwen3-0.6B.

7.2 Held-out relevance and BFCL irrelevance

Both Granite models decline only 2–4% of the requests in our relevance set that need no call, but 55–67% on BFCL irrelevance. The two sets differ in what the tools look like:

- **Our decline requests** sit in the same app domain as the offered tools. For example, "turn bluetooth off please" comes with Wi-Fi, mobile-data, volume and hotspot tools, which tempts a model to call the closest one.
- **BFCL's irrelevance prompts** usually offer a single function that does a different calculation or lookup.

Udgam-384M-Chat was trained on relevance examples in the same style as ours, which favours it there. On BFCL irrelevance it is in the middle of the group.

7.3 Documents and shell commands

- **Document → JSON.** Every output parses as JSON, and 56% are valid under the unseen schema. The field-level F1 is 68.7, but only 1 of 50 documents has every field exactly right. Validate against the schema before use.
- **Shell.** The model chooses the right kind of answer for 85.9% of tasks and writes valid bash 96.6% of the time. But only 2 of the 26 commands that are run (7.7%) produce the right output; every model here scores 0–8% on this check. Shell commands must be confirmed by a person before they run.

7.4 Temperature costs tool accuracy

We measured sampling presets in Ollama on 9 October 2026, with Q8_0 and thinking on (Ollama's default), using 2 seeds per sampled preset:

PRESET	HELD-OUT APP ACTIONS: EXACT CALL	BFCL V3 SIMPLE	CHAT ANSWERS THAT HIT THE LENGTH CAP	IFEVAL STRICT (80 PROMPTS)
Greedy	24.3	60.0	30%	25.0
Temperature 0.7, top-p 0.95, top-k 64	20.0	46.3	7%	28.1
Same, repeat penalty 1.1	20.0	46.7	2%	25.0
Temperature 0.5, top-p 0.9, top-k 40, repeat penalty 1.1	22.1	52.3	2.5%	30.6
Temperature 0.3, same	19.3	55.3	5%	28.8
Temperature 0.1, same (default)	21.4	59.3	8%	30.6

- **Temperature.** BFCL simple falls steadily as the temperature rises.
- **Repeat penalty.** It costs no tool accuracy and removes the repetition loops ("offline, offline, offline") and most runaway answers.
- **Shell and relevance.** A separate check found no preset that clearly changed behaviour: 80–84% right shell behaviour and 65–72% relevance.

7.5 Thinking mode does not help

For the f16 build, turning thinking on:

- lowers held-out exact calls from 32.9% to 25.7%;
- lowers relevance decisions from 76.7% to 63.3%;
- raises document F1 from 68.8 to 71.2;
- leaves BFCL and IFEval within a point.

7.6 What post-training changed

TEST (HARNESS OF SECTION 6.1, F16)	UDGAM-384M-BASE-4K	UDGAM-384M-CHAT
BFCL v3 simple / multiple / parallel	66.0 / 75.0 / 48.0	60 / 67 / 50
BFCL v3 irrelevance	11.3	70
Held-out app actions: exact call	24.3	32.9
Held-out relevance: right decision	73.3	76.7
Held-out document → JSON: schema-valid	48	58
Held-out shell: right behaviour	78.1	85.9
IFEval prompt-level strict	15.9	26.2

Post-training turned a model that calls a tool for almost every BFCL irrelevance prompt into one that declines 70% of them, and it raised every held-out score and IFEval. The cost was 6 to 8 points on BFCL simple and multiple, where the base model, which saw tool calls in pretraining, was already calling tools whenever possible.

7.7 What a missing or doubled `<bos>` costs

On 17 October 2026 we scored 200 documents (100,822 tokens, from a held-out validation shard of a later Udgam Labs data build) with the released base weights. We used transformers (fp32, Apple GPU) and put 0, 1 or 2 `<bos>` tokens in front of each document:

LEADING <code><BOS></code> TOKENS	BASE, NATS PER TOKEN	BASE-4K, NATS PER TOKEN	GREEDY CONTINUATION OF "THE CAPITAL OF FRANCE IS"
0	2.610 (+26.0%)	2.617 (+26.1%)	" is is is is is ..."
1 (as trained)	2.072	2.075	" Paris. The capital of France is Paris. ..."
2	2.081 (+0.44%)	2.084 (+0.43%)	" the capital of the French Republic, and ..."

Without `<bos>`, the base models degenerate. A doubled `<bos>` costs little on average, but it changes outputs.

For the chat model, we repeated the re-run of section 6.2 with a second `<bos>` before every prompt:

- 814 of the 1,265 outputs changed.
- BFCL simple fell from 60.0% to 56.7%, multiple from 67% to 60%, and parallel from 46% to 44%.
- BFCL irrelevance rose from 68% to 72.7%.
- Held-out exact calls stayed at 31.4%.

Use exactly one `<bos>` (section 5.3).

8. Limitations and safety

8.1 Limitations

- **Unseen tools.** About one call in three is exactly right. The model sometimes picks the wrong tool, and it usually guesses a missing value instead of asking.
- **Shell.** Commands are usually valid syntax but rarely correct on unfamiliar tasks, and destructive requests are not reliably refused.
- **Instruction following.** IFEval prompt-level strict is 26.1%. 37% of its IFEval answers ran to the 768-token limit, against 5–10% for the Granite models.
- **Knowledge.** The model is fluent but often wrong on facts, arithmetic and unit conversions. Asked for the boiling point of water in Fahrenheit, it answered "about 37 degrees Celsius, roughly 100 degrees Fahrenheit". It is not suitable for medical, legal or financial advice, or for unsupervised open-ended chat.
- **Scope.** English only. The context is 4,096 tokens (2,048 for Udgam-384M-Base), and longer inputs fail.
- **Evaluation.**
 - The held-out sets are our own, small and favour this model.
 - Our BFCL subsets and checker are not the official leaderboard.
 - The results come from one training run.
 - The speed figures come from one shared Mac.
- **Machine-generated data.** Part of the training text was machine-generated, and its errors and habits can carry over.
- **Base models.** They have no instruction or safety tuning, and they will continue any text. Prompted as chatbots, they ramble in a "thinking aloud" style. Do not deploy them directly to end users.

8.2 Safety

Udgam-384M-Chat was fine-tuned with examples of declining harmful requests and of helping with benign requests that only sound alarming. The result is below what an open chatbot needs, and we release the model as **not safety-hardened**.

XSTest. We ran the Q8_0 build in llama.cpp (greedy, thinking off) and scored it with the refusal-phrase classifier. It declines 54.0% of the 200 unsafe prompts and over-refuses 14.8% of the

250 safe ones. Q4_K_M gives the same numbers, and f16 gives 53.5% and 14.4%.

UNSAFE PROMPT TYPE (25 EACH)	DECLINED	SAFE PROMPT TYPE (25 EACH)	REFUSED
Figurative language	23	Figurative language ("kill time")	13
Safe targets	20	Homonyms ("shoot a photo")	8
Definitions	18	Safe targets	8
Safe contexts	18	Privacy, fictional people	4
Homonyms	15	Safe contexts	4
Historical events	8	Definitions, historical events, privacy of public figures	0 each
Privacy	6	Nonsense group or nonsense discrimination	0 each
Discrimination	0		
All 200	54.0%	All 250	14.8%

The model never declines the discriminatory-claim prompts and rarely the privacy and historical-atrocity prompts. It over-refuses idioms and homonyms.

Other measures. An LLM judge (gpt-oss-20b [16]) answered two questions per reply: does it refuse explicitly, and does it attempt the request. The model was the f16 weights in transformers (fp32, greedy, up to 256 new tokens), with thinking off / on:

TEST	DECLINED OR REFUSED
XSTest, 200 unsafe prompts	56.5% / 57.0%
XSTest, 250 safe prompts	14.4% / 15.6%
Aegis 2.0 test split [17], 300 unsafe prompts	47.3% / 47.7%
Aegis 2.0 test split, 200 safe prompts	17.0% / 16.0%
OR-Bench-Hard [18], 200 safe prompts that sound harmful	4.5% / 4.0%

Compared with other models, the Granite models decline more unsafe XSTest prompts (80–93%) but also refuse 37–45% of safe ones. Qwen3-0.6B and SmoLLM2 decline only 17% and 28% of unsafe prompts.

Recommendations:

- Offer only the tools the user may use, validate every argument, and confirm consequential actions (send, pay, delete, share, run) with the user.
- Never run shell commands automatically.

- Put a moderation model or policy layer in front of any user-facing open chat.
- Do not deploy the base models to end users.

9. Release details

ITEM	WHERE
Chat model: weights and model card	huggingface.co/udgamlabs/Udgam-384M-Chat
GGUF builds and Ollama Modelfiles	huggingface.co/udgamlabs/Udgam-384M-Chat-GGUF
MLX builds	Udgam-384M-Chat-MLX , Udgam-384M-Chat-MLX-4bit
Base models	Udgam-384M-Base , Udgam-384M-Base-4K
Documentation, examples, checksums (v1.0.0)	github.com/udgamlabs/udgam-384m
Walkthrough page	udgamlabs.github.io/udgam-384m
Model page	udgamlabs.com/models/udgam-384m

- **Release:** 10 October 2026, version 1.0.0.
- **Tokenizer update:** on 12 October 2026, the `tokenizer.json` files of the five transformers and MLX repositories were updated so that plain text gets one `<bos>`, as in training and in the GGUF builds (section 5.3). `SHA256SUMS` in the GitHub repository lists the files as of 10 October.
- **Requirements:**
 - transformers 5.18 or newer;
 - a llama.cpp build with Gemma 4 support (we tested d7a695e);
 - Ollama with the `gemma4` renderer;
 - mlx-lm 0.32 or newer.
- **Licence:** Apache-2.0 for the weights, tokenizer, documentation and examples. The chat template is the official Gemma 4 template (Apache-2.0).
- **Contact:** the Community tab of the Hugging Face model pages, or issues on GitHub.

```
@misc{udgam384m2026,  
  title = {Udgam-384M: a small on-device tool-calling model trained from  
scratch},  
  author = {{Udgam Labs}},  
  year = {2026},  
  url = {https://huggingface.co/udgamlabs/Udgam-384M-Chat}  
}
```

10. Conclusion

A 384M-parameter model trained from scratch on roughly 8B tokens can serve as the decision layer of an app. On held-out tools, documents and shell tasks, it decides whether to call, ask, decline or refuse more reliably than four other small open models, and it returns schema-valid JSON more often. In our release-time measurement it also generated text fastest of the five on an Apple-silicon Mac.

It is not an accurate tool caller on its own:

- **Neutral benchmarks.** It is behind Granite 4.0 350M and Qwen3-0.6B: 60% against 80–91% on BFCL v3 simple, and 26% against 47–58% on IFEval. IBM reports about 1,900 times more pretraining tokens for Granite 4.0 350M.
- **Unseen tools.** Two calls in three need correcting.
- **Other weaknesses.** It chooses the wrong tool on unfamiliar tools, guesses instead of asking, writes shell commands that run but give the wrong result, and its safety behaviour is weak.

More varied tool definitions in training and execution-checked training for shell commands are the next levers. The weights, builds and examples are open under Apache-2.0, so the results can be checked and the models can be fine-tuned.

11. References

1. Gemma Team (2026). *Gemma 4 Technical Report*. [arXiv:2607.02770](https://arxiv.org/abs/2607.02770). Model card: huggingface.co/google/gemma-4-E2B-it.
2. Patil, S. G. et al. (2025). *The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models*. ICML 2025. Leaderboard and data: gorilla.cs.berkeley.edu/leaderboard.html.
3. Patil, S. G., Zhang, T., Wang, X. and Gonzalez, J. E. (2023). *Gorilla: Large Language Model Connected with Massive APIs*. [arXiv:2305.15334](https://arxiv.org/abs/2305.15334).

4. Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D. and Hou, L. (2023). *Instruction-Following Evaluation for Large Language Models*. [arXiv:2311.07911](#).
5. Röttger, P., Kirk, H. R., Vidgen, B., Attanasio, G., Bianchi, F. and Hovy, D. (2023). *XSTest: A Test Suite for Identifying Exaggerated Safety Behaviours in Large Language Models*. NAACL 2024; [arXiv:2308.01263](#).
6. ggml-org. *llama.cpp* and the GGUF format. [github.com/ggml-org/llama.cpp](#); [github.com/ggml-org/ggml/blob/master/docs/gguf.md](#).
7. Ollama. [github.com/ollama/ollama](#).
8. Apple machine learning research. *MLX* and *mlx-lm*. [github.com/ml-explore/mlx](#); [github.com/ml-explore/mlx-lm](#).
9. Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F. and Sanghai, S. (2023). *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. [arXiv:2305.13245](#).
10. Beltagy, I., Peters, M. E. and Cohan, A. (2020). *Longformer: The Long-Document Transformer*. [arXiv:2004.05150](#).
11. Gemma Team (2024). *Gemma 2: Improving Open Language Models at a Practical Size*. [arXiv:2408.00118](#).
12. Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B. and Liu, Y. (2021). *RoFormer: Enhanced Transformer with Rotary Position Embedding*. [arXiv:2104.09864](#).
13. Shazeer, N. (2020). *GLU Variants Improve Transformer*. [arXiv:2002.05202](#).
14. Zhang, B. and Sennrich, R. (2019). *Root Mean Square Layer Normalization*. [arXiv:1910.07467](#).
15. Henry, A., Dachapally, P. R., Pawar, S. and Chen, Y. (2020). *Query-Key Normalization for Transformers*. [arXiv:2010.04245](#).
16. OpenAI (2025). *gpt-oss-120b & gpt-oss-20b Model Card*. [arXiv:2508.10925](#).
17. Ghosh, S. et al. (2025). *Aegis2.0: A Diverse AI Safety Dataset and Risks Taxonomy for Alignment of LLM Guardrails*. [arXiv:2501.09004](#).
18. Cui, J., Chiang, W.-L., Stoica, I. and Hsieh, C.-J. (2024). *OR-Bench: An Over-Refusal Benchmark for Large Language Models*. [arXiv:2405.20947](#).
19. Qwen Team (2025). *Qwen3 Technical Report*. [arXiv:2505.09388](#). Model card: [huggingface.co/Qwen/Qwen3-0.6B](#).
20. EleutherAI. *lm-evaluation-harness*. [github.com/EleutherAI/lm-evaluation-harness](#).
21. IBM. *Granite 4.0 Nano* model cards: [granite-4.0-350m](#), [granite-4.0-h-350m](#) and [granite-4.0-350m-base](#), Hugging Face (accessed 17 October 2026).

22. Hugging Face. *SmolLM2-360M-Instruct* model card.
huggingface.co/HuggingFaceTB/SmolLM2-360M-Instruct (accessed 17 October 2026).
23. Udgam Labs. Udgam-384M model cards, GitHub repository (`docs/BENCHMARKS.md` ,
examples and `SHA256SUMS`) and walkthrough page, October 2026.

About Udgam Labs. Udgam Labs is an independent AI research lab. We work from first principles: before we build, we understand why things work, whether we are training a model, fine-tuning one or applying AI to a new problem. Udgam (उद्गम) is Sanskrit for origin. udgamlabs.com