
# AI Developer Cheat-Sheet

**Light system. Heavy thinking.** A one-stop quick reference for building with AI - locally and with cloud agents. Prepare your machine, pick the right model for the job, validate your work, and ship. Part of **local-ai-lab** by Nik Reljin.

> **Model names move fast.** Cloud/media/local model names below are current **as of June 2026** and every row links to the source - always confirm the exact version at the link. Claude model IDs are exact and stable.

---

## 1 · Prepare your local (Ollama + BitNet.cpp)

Run models on your own hardware - no API keys, no data leaving the box. Ollama for general local LLMs; BitNet.cpp for ultra-light 1-bit models that run on CPU / older machines.

### Ollama - ollama.com

| Step | Command |
|---|---|
| Install (Linux) | `curl -fsSL https://ollama.com/install.sh \| sh` |
| Start the server | `ollama serve` **(API on** `http://localhost:11434`**)** |
| Pull a model | `ollama pull qwen2.5-coder:7b` |
| Chat in terminal | `ollama run llama3.3` |
| List / remove | `ollama list` · `ollama rm <model>` |
| Use the API | `curl http://localhost:11434/api/generate -d '{"model":"llama3.3","prompt":"hi"}'` |
| GPU / tuning | `OLLAMA_NUM_GPU`, `OLLAMA_HOST`, `OLLAMA_KEEP_ALIVE` env vars |

**Pick-and-run a model from a simple UI with ai-runner.**

### BitNet.cpp - github.com/microsoft/BitNet

1-bit (ternary -1/0/+1) inference, ~0.4 GB RAM for a 2B model, CPU-friendly. Based on `llama.cpp`.

| Step | Command |
|---|---|
| Clone + submodules | `git clone --recursive https://github.com/microsoft/BitNet && cd BitNet` |
| Build | `pip install -r requirements.txt && python setup_env.py -md models -q i2_s` |
| Get a model | `huggingface-cli download microsoft/bitnet-b1.58-2B-4T-gguf --local-dir models/BitNet-b1.58-2B-4T` |
| Run inference | `python run_inference.py -m models/.../ggml-model-i2_s.gguf -p "Hello" -cnv` |

**Producing your own small models? See shrink-llm (quantize/prune/distill) and finetorch (LoRA/QLoRA).**

---

## 2 · Coding agents · testing · validating PRs

### Pick a coding agent

| Agent | Best at | Local models? | Notes |
|---|---|---|---|
| **Claude Code** | Long-horizon agentic coding, refactors, reviews | Cloud (Claude); local via proxy | Default `claude-opus-4-8`; `/code-review`, `/security-review` built in |
| **OpenAI Codex (CLI)** | Agentic "Goal Mode", PR creation, frontend | Cloud (GPT-5.x) | Runs on GPT-5.5 / GPT-5.2-Codex; MCP support |
| **Gemini CLI** | Large-context, vision-assisted, free tier | Cloud (Gemini 2.5) | Strong on huge context + image input |
| **Ollama + your editor** | Fully offline, private | **Yes** - `qwen2.5-coder`, `llama3.3` | No keys, no egress; pair with Continue/aider |

### Run the tests (this repo's pattern)

```
python -m venv venv && source venv/bin/activate
pip install -r requirements.txt
pytest -q                # offline tests - no network, no LLM
./run -l N test          # run a lesson's tests via the harness
```

- Keep unit tests **offline** (no network, no live model) so they're fast and deterministic.
- Let an agent **write and run** the tests, then read the failures - don't trust "looks right".

### PR validation checklist

- `[ ]` Tests green locally (`pytest -q`) and in CI
- `[ ]` **Conventional Commits** - `feat:` `fix:` `chore:` `docs:` `refactor:` `ci:`
- `[ ]` Ran `/code-review` (bugs) and `/security-review` (secrets, injection) on the diff
- `[ ]` No secrets committed - `.env` ignored, placeholders in `.env.example`
- `[ ]` CHANGELOG updated if the release workflow keys off it
- `[ ]` **No AI-attribution trailers** - author is you; no "Co-Authored-By: ...AI", no "Generated with..."

---

## 3 · Which model for what (the picker)

Match the **task** to a **cloud** pick and a **local** equivalent. Cloud = best quality / least setup; local = private / free / offline.

### Text - code, chat, reasoning

| Task | Cloud (service · model) | Local (Ollama / BitNet) |
|---|---|---|
| Agentic coding | **Claude** `claude-opus-4-8` · docs | `qwen2.5-coder:32b` |
| Fast / cheap code & chat | **Claude** `claude-sonnet-4-6` / `claude-haiku-4-5`; **Gemini** 2.5 Flash | `qwen2.5-coder:7b`, `llama3.3:8b` |
| Hardest reasoning | **Claude** `claude-fable-5`; **OpenAI** GPT-5.5; **Gemini** 2.5 Pro | `llama3.3:70b` |
| CPU-only / tiny footprint | - | **BitNet** `bitnet-b1.58-2B-4T` |

### Vision - analyze / describe images

| Task | Cloud | Local |
|---|---|---|
| Read content in an image, charts, docs | **Gemini** 2.5 Pro/Flash (vision); **Claude** vision (`claude-opus-4-8`) | `qwen2.5-vl`, `llama3.2-vision`, `gemma3` |
| OCR / document extraction | Gemini 2.5 Pro; Claude (PDF input) | `qwen2.5-vl` + Tesseract |

### Embeddings - for RAG / search

| Task | Cloud | Local |
|---|---|---|
| Embed text for retrieval | OpenAI `text-embedding-3`; Gemini embeddings | `nomic-embed-text` (default), `bge-m3` |

### Media generation (links - verify versions)

| Make... | Service · model | Link |
|---|---|---|

| | | |
|---|---|---|
| **Images** | Gemini 2.5 Flash Image ("Nano Banana") · FLUX.2 · GPT-Image · Imagen 4 | nano-banana · FLUX |
| Images (local) | **SDXL** / FLUX via ComfyUI / Diffusers | ComfyUI |
| **Video** | Veo 3.1 (audio) · Kling 3.0 (cheap) · Runway Gen-4.5 (control) · Seedance 2.0 | Veo · Runway |
| **Speech → text (STT)** | OpenAI Whisper API · Gemini | **Whisper** (local), `faster-whisper` |
| **Text → speech (TTS)** | ElevenLabs · Gemini TTS | **Kokoro**, **Piper** (local) |

> Default to the **latest Claude** model for serious code/agent work; reach for Gemini for big-context + vision, and local models when privacy/offline/cost matters.

---

## 4 · RAG - do / don't

Pipeline (see `localrag/`): **extract → chunk → store/embed → retrieve → generate (grounded)**.

| Do | Don't |
|---|---|
| Right-size chunks; keep overlap small and consistent | Dump the whole corpus into the prompt |
| Retrieve top-k, then **cite** sources in the answer | Answer ungrounded / without showing sources |
| Use **hybrid** (BM25 + embeddings) for recall **(Lesson 3)** | Rely on a single retrieval signal |
| **Evaluate** retrieval + answers, regression-test **(Lesson 5)** | Ship without an eval / "it looked fine" |
| Treat retrieved text as **data**, sanitize it **(Lesson 4)** | Treat retrieved text as instructions (injection) |
| **Prompt-cache** large stable context to cut cost/latency | Re-send the same big context uncached every call |

---

## 5 · MCP & tool design - do / don't

Expose your retriever/tools to an LLM via **MCP** (see `mcp_server.py` - FastMCP over stdio). Claude can also connect to MCP servers via the `mcp_servers` API param.

| Do | Don't |
|---|---|
| Clear names; **prescriptive descriptions** ("call this when...") | Vague names / what-it-does-only descriptions |

| | |
|---|---|
| Typed JSON schema; `enum` for fixed sets; mark `required` | Untyped free-form string inputs |
| Return errors with `is_error: true` + a useful message | Swallow errors or return silent failures |
| Keep the tool set **small and focused** | Expose dozens of overlapping tools |
| **Gate** side-effecting / irreversible tools | Auto-run destructive actions unprompted |
| Keep secrets server-side; pass via env/vault | Put API keys in prompts, args, or tool output |

---

## 6 · LangChain / LangGraph / Lang* - when to use what

| Tool | Use it for | Don't |
|---|---|---|
| **LangChain** | Glue: chains, loaders, retrievers, model adapters | Wrapping a single one-shot call in 5 abstractions |
| **LangGraph** | Stateful agents / multi-step graphs, retries, branching | Linear pipelines a plain function handles |
| **LangSmith** | Tracing, eval, debugging prod chains | As a hard dependency for tiny scripts |
| **LlamaIndex** | Heavy RAG indexing / query engines | When `localrag` + a vector store already fits |

All of the above point at **local Ollama** via the OpenAI-compatible endpoint (`http://localhost:11434/v1`) - develop offline, swap to cloud later. **Don't over-abstract**: if it's one model call, just call the model.

---

## 7 · Shrinking LLMs for local use

Make big models fit small machines. See shrink-llm.

| Technique | What it does | Trade-off |
|---|---|---|
| **Quantization** (GGUF `q4`/`q5`/`q8`) | Fewer bits per weight | `q4` ≈ 4× smaller, slight quality drop; `q8` near-lossless |
| **Pruning** | Drop low-impact weights | Needs fine-tune to recover quality |
| **Distillation** | Train a small "student" from a big "teacher" | Up-front training cost |
| **1-bit (BitNet)** | Native ternary weights | Smallest/fastest on CPU; use the native b1.58 models |

Rule of thumb: start with a `q4`/`q5` **GGUF** of a good model in Ollama; drop to **BitNet** when you need CPU-only or sub-GB footprint; reach for pruning/distillation/LoRA only when a stock model won't fit or won't perform.

---

## Links

- **local-ai-lab** course & code - https://github.com/nikolareljin/local-ai-lab
- Ollama - https://ollama.com · Models - https://ollama.com/library
- BitNet - https://github.com/microsoft/BitNet
- Model Context Protocol - https://modelcontextprotocol.io
- Claude Platform docs - https://platform.claude.com/docs
- Google AI (Gemini) - https://ai.google.dev
- shrink-llm - https://github.com/nikolareljin/shrink-llm · finetorch - https://github.com/nikolareljin/finetorch · ai-runner - https://github.com/nikolareljin/ai-runner

---

**local-ai·lab - built and maintained by Nik Reljin. Cheat-sheet current as of June 2026; verify fast-moving model versions at the linked sources.**