Model Routing for Local AI — Stop Using One Model for Everything

February 25, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Keep 2-3 models loaded in Ollama: a small one (Qwen 2.5 3B) for quick tasks, a medium one (Qwen 2.5 14B) for daily work, and a large one (32B) for complex reasoning. Route manually by picking the right model per task, or automate it with LiteLLM's built-in complexity router. The cost difference between using 3B and 32B for a simple question is 10x in speed and VRAM. For cloud tasks, Gemini Flash-Lite costs $0.10/M tokens vs Pro at $2.00/M — 20x difference.

You're probably running Qwen 32B for everything. Summarizing emails, writing code, answering quick questions, analyzing documents. That's like driving a semi truck to buy milk.

A 32B model uses 20GB+ of VRAM, generates maybe 15-20 tokens per second, and occupies your entire GPU. Meanwhile half your tasks would get identical results from a 3B model running at 80+ tokens per second on 2GB of VRAM.

Model routing means sending each task to the right model at the right cost. It's the most undermeasured skill in local AI and the single biggest efficiency gain most people ignore.

The one-model trap

Most local AI users pick one model and use it for everything. The "best model" lists reinforce this: find the top scorer, download it, done. But no single model is best at everything, and even good models waste resources on tasks that don't need them.

The cost isn't just API tokens. Locally, you're paying in VRAM, generation speed, context window budget, and electricity. A 32B model answering "what time zone is Tokyo in?" is burning 20 watts to do a job that a 3B model handles in a tenth of the time. Multiply that across hundreds of queries a day and the waste adds up.

37% of enterprises now run 5+ models in production. IDC forecasts that by 2028, 70% of top AI-driven companies will use dynamic multi-model routing. This isn't a future optimization. It's standard practice that local AI users haven't adopted yet.

The task scope axis

Every task you throw at an AI sits somewhere on a spectrum from narrow to wide.

On the narrow end: classify this email, extract a date, translate a sentence, yes/no questions. Clear, bounded answers. A 3B model handles them instantly with minimal VRAM. Running a 32B model here is like hiring a surgeon to apply a Band-Aid.

In the middle: summarize this document, write a function, answer a research question, compare two options. These need some reasoning and world knowledge but have well-defined scope. A 14B model hits the sweet spot.

On the wide end: multi-step research across unfamiliar domains, debugging across a codebase, creative writing with a specific voice. These need representational depth that smaller models don't have.

At the far end, agentic work: sustained multi-hour workflows, tool orchestration with error recovery, novel problem-solving where the model needs to rethink strategy when things break. Distilled models fall apart here. Frontier models (Claude Opus, GPT-5, Gemini Pro) are still measurably better for this class of work.

Knowing where your task sits on this spectrum is the entire routing skill.

How to implement routing locally

Four strategies, from simplest to most automated.

Manual routing: just pick the model

Keep 2-3 models in Ollama. Pick the right one for each task.

- Small (Qwen 2.5 3B or Phi-4 Mini 3.8B) for quick lookups, classification, simple chat
- Medium (Qwen 2.5 14B or Llama 3.1 8B) for daily coding, summarization, general work
- Large (Qwen 2.5 32B or DeepSeek-Coder 33B) for complex reasoning, architecture decisions, research

This is the approach most people should start with. No infrastructure, no code, just discipline about choosing the right tool. Ollama loads models on demand and you can set `OLLAMA_KEEP_ALIVE` to control how long each stays in memory.

Open WebUI model switching

Open WebUI lets you switch models mid-conversation with a dropdown. Start with a small model for initial Q&A, switch to a larger one when complexity increases. Type `@modelname` to invoke a specific model inline, or select two models simultaneously and compare their responses side by side.

No code required. You can set up custom model presets with different system prompts, temperature settings, and tools per profile. A “quick assistant” preset pointing to your 3B model and a “deep work” preset pointing to 32B makes routing a dropdown choice.

Automated routing with LiteLLM

LiteLLM is a proxy server that sits between your applications and your models, routing requests automatically. It speaks the OpenAI API format, so anything that works with ChatGPT works with LiteLLM pointed at your local models.

The built-in complexity router scores requests across seven dimensions (token count, code presence, reasoning markers, technical terminology) and routes to the right tier with sub-millisecond overhead:

```
# litellm_config.yaml
model_list:
  - model_name: quick
    litellm_params:
      model: ollama_chat/qwen2.5:3b
      api_base: http://localhost:11434
  - model_name: balanced
    litellm_params:
      model: ollama_chat/qwen2.5:14b
      api_base: http://localhost:11434
  - model_name: heavy
    litellm_params:
      model: ollama_chat/qwen2.5:32b
      api_base: http://localhost:11434
  - model_name: frontier
    litellm_params:
      model: anthropic/claude-sonnet-4-5-20250929
      api_key: os.environ/ANTHROPIC_API_KEY

complexity_router_config:
  tiers:
    SIMPLE: quick
    MEDIUM: balanced
```

```
    COMPLEX: heavy
    REASONING: frontier
  tier_boundaries:
    simple_medium: 0.15
    medium_complex: 0.35
    complex_reasoning: 0.60
```

Start with `litellm --config litellm_config.yaml` and point your apps at `http://localhost:4000`. Every request gets scored and routed automatically. LiteLLM handles 1,500+ requests per second.

You can also set up fallbacks so if a local model fails, the request falls through to a cloud model:

```
litellm_settings:
  fallbacks: [{"heavy": ["frontier"]}]
  context_window_fallbacks:
    - {"balanced": ["heavy"]}
```

Hybrid local + cloud routing

The privacy-aware version: anything with personal data stays local, everything else goes to whichever model gives the best results.

LiteLLM makes this transparent. Your application code doesn't know whether it's talking to Ollama or Claude. You configure the routing rules once, and every request goes to the right place. A simple Python router handles the privacy dimension:

```
import litellm

def route_request(prompt, contains_private_data=False):
    if contains_private_data:
        # Private data never leaves your machine
        return litellm.completion(
            model="ollama_chat/qwen2.5:14b",
            messages=[{"role": "user", "content": prompt}]
        )
    else:
        # Non-sensitive tasks get the best available model
        return litellm.completion(
            model="anthropic/claude-sonnet-4-5-20250929",
```

```
            messages=[{"role": "user", "content": prompt}]
        )
```

The VRAM budget approach

You have X GB of VRAM. How do you spend it?

Strategy	Models	VRAM Used	Trade-off
One big model	32B Q4_K_M	~20GB	Good at everything, great at nothing specific
Two models	7B Q8 + 7B embedding	~12GB	Chat + RAG simultaneously
Three models	3B + 7B + swap in 32B	~6GB active, 20GB peak	Best routing, needs model swapping

Option C is the routing approach. Keep the 3B and 7B loaded permanently (they fit together in 6GB). When a complex task comes in, the 7B unloads and the 32B loads. Ollama handles this automatically: set `OLLAMA_MAX_LOADED_MODELS=2` and `OLLAMA_KEEP_ALIVE=10m`, and idle models unload after 10 minutes to free VRAM.

The math for simultaneous models:

```
Required VRAM = model_size + (num_parallel × context_length × overhead)
```

A 7B Q4 model with 4 parallel requests at 2K context needs roughly 6GB total. Two 7B models running simultaneously need ~12GB. Check exact numbers with the VRAM Calculator and our VRAM requirements guide.

With 24GB (RTX 3090 / 4090), you can comfortably keep a 3B and a 14B loaded simultaneously, or run a single 32B at Q4 quantization. The routing question becomes: do you want two fast models covering different tasks, or one versatile model that's slower at everything?

Routing for agent swarms

Multi-agent systems like CrewAI and AutoGen let you assign different models to different agents. This is where routing really pays off.

A typical agent swarm has agents doing very different jobs. The research agent needs reasoning depth and long context coherence, so it gets 32B or a frontier API. The data extraction agent runs a narrow, well-defined task where 7B is fine. Summarization gets 14B. The coordinator needs broad understanding and tool orchestration, so it gets 32B or frontier.

Most frameworks default every agent to the same model. That's wasteful. If your swarm makes 100 calls per run and 70 of those are simple extraction or summarization, you've cut your compute budget by 70% on those calls with zero quality loss by routing them to smaller models. The swarm itself becomes the routing layer. See our local AI agents guide for setup details.

When to route to cloud

Local models in 2026 are genuinely good for 80% of tasks. The remaining 20% is where frontier models are still measurably better, and being honest about this saves you time.

Tasks where frontier still wins:

- Multi-file code refactoring across large codebases
- Long-form writing that needs to maintain a specific voice across thousands of words
- Multi-step research that requires synthesizing information from many sources
- Tasks needing very long coherent context (50K+ tokens)
- Sustained agentic work where the model needs to recover from unexpected failures

Google's pricing shows how the cloud tier strategy works. Gemini Flash-Lite costs $0.10 per million input tokens. Gemini Pro costs $2.00. That's a 20x difference. Google themselves route their own products to different tiers based on task complexity.

Local-first for privacy and cost, cloud fallback for capability. LiteLLM's fallback configuration handles this automatically. Complex tasks go to frontier models only when local models can't handle them.

This gap is closing. Every six months, the tasks that require frontier shrink. But it's not closed yet. Pretending your 14B local model matches Claude Opus on complex reasoning doesn't help anyone. Know where the limits are and route around them.

The model routing cheat sheet

Task	Model Tier	Local Pick	Why
	Small (1-3B)	Qwen 2.5 3B, Phi-4 Mini	Speed, minimal VRAM

Task	Model Tier	Local Pick	Why
Quick Q&A, classification			
Email, simple chat	Small-Med (7-8B)	Llama 3.1 8B, Mistral 7B	Good enough, fast
Code completion	Medium (14B)	Qwen 2.5 Coder 14B	Needs code understanding
Document analysis	Medium-Large (14-32B)	Qwen 2.5 14B or 32B	Context + reasoning
Research synthesis	Large (32B+)	Qwen 2.5 32B, DeepSeek 33B	Multiple sources, judgment calls
Multi-step agents	Large or Frontier	32B local or Claude API	Manifold width matters
Creative writing	Large or Frontier	32B+ or Claude API	Voice, coherence, subtlety

Start with manual routing between two models. Most people see immediate improvement just from stopping the habit of throwing every question at their biggest model. Once that feels natural, set up LiteLLM to automate it.

The best coding models don't overlap with the best chat models. The best chat models aren't the fastest at classification. Matching model to task is how you get more from the same hardware. Your 24GB GPU is capable of a lot more than running one model at a time.
