OPENJARVIS: PERSONAL AI, ON PERSONAL DEVICES

Anonymous authors
Paper under double-blind review

ABSTRACT

Personal AI stacks for writing, research, coding, and scheduling are becoming central to daily work, yet most still route every query to cloud-hosted frontier models. This paradigm exposes private data and creates recurring API spend. Excitingly, open-source models and consumer accelerators are significantly improving, enabling the possibility of a personal AI stack running on-device. However, we find that replacing cloud models with local models in existing personal AI stacks, such as OpenClaw and Hermes Agent, leads to significant deterioration in performance. For example, replacing Claude Opus 4.6 with Qwen3.5-9B drops accuracy by 25–39 percentage points across PinchBench and GAIA. This collapse reflects both a model-capability gap and harness incompatibility: agentic prompts, tool descriptions, memory configuration, and inference runtime settings carry model-specific defaults that materially contribute to the substitution gap. Towards building an on-device personal AI stack, we present OPENJARVIS, an architecture that represents a personal AI system as a typed *spec* over five primitives: Intelligence, Engine, Agents, Tools & Memory, and Learning. By exposing each primitive as an independently editable field, the *spec* makes the stack portable, measurable, and end-to-end optimizable around any choice of model. On-device *spec*s match or exceed cloud accuracy on 4 out of 8 benchmarks, and the best one (Qwen3.5-122B) lands within 3.2 pp of the best cloud baseline on average with up to a $\sim$6,600$\times$ reduction in marginal API cost; Qwen3.5-35B achieves 4$\times$ lower end-to-end latency. To close the remaining accuracy gap between the best cloud model and the best local model, OPENJARVIS uses *LLM-guided spec search*, in which a frontier teacher proposes edits across the *spec* and a held-out gate accepts only non-regressing improvements. Search improves local students by 13–32 pp on average at 7–11$\times$ lower optimization cost than LoRA fine-tuning, with student inference and training on-device.

1 INTRODUCTION

Millions of users now rely on personal AI stacks like OpenClaw (Steinberger & OpenClaw Contributors, 2025) and Hermes Agent (Nous Research, 2026) for writing, research, coding, and scheduling (Malik, 2026). Yet today's personal AI is rarely private or local: most systems route each query to a cloud-hosted frontier model. That design exposes private data to third-party servers, creates thousands of dollars per year in recurring API spend (Gartner, Inc., 2026), and consumes orders of magnitude more energy per token than local execution (Saad-Falcon et al., 2025). This paper asks whether the core of a personal AI stack (model inference, agent execution, memory, telemetry, and learning) can run *on-device*, while still being competitive with cloud-only stacks.

The components for an on-device stack already exist. Consumer neural accelerators (Apple Inc., 2024; Qualcomm Technologies, 2025; Intel Corporation, 2023) run 1B–128B open-weight models with FP8 or FP4 quantization (Gerganov, 2023). Intelligence delivered per watt by local AI has improved 5.3$\times$ from 2023 to 2025 (Saad-Falcon et al., 2025). Open-weight families such as Qwen3.5 (Qwen Team, 2026) and Gemma (Farabet & Lacombe, 2026) now trail frontier cloud models by 6–12 months, rather than years, on personal AI tasks (Chiang et al., 2024; Saad-Falcon et al., 2025). All of these components are necessary but not sufficient. We find that existing personal AI stacks are usually tuned to specific cloud models: agent prompts and reasoning loops, tool descriptions, memory configuration, and engine/runtime settings. Naive substitution exposes this coupling: replacing `Claude Opus 4.6` with `Qwen3.5-9B`, while keeping the rest of OpenClaw or Hermes Agent

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

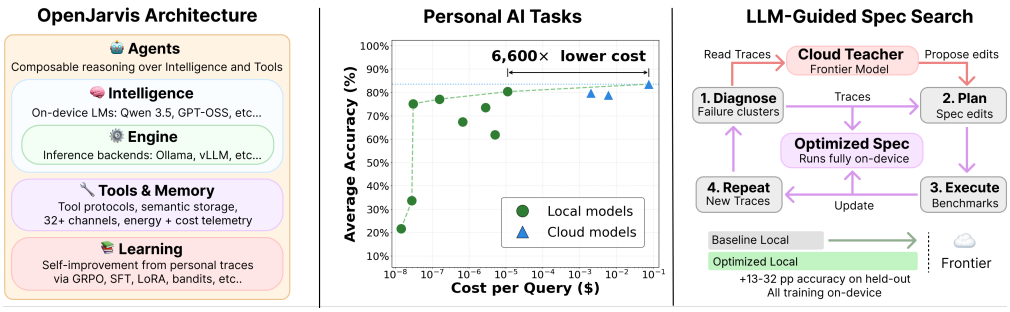


Figure 1: **Overview of OpenJarvis. (Left)** Five composable primitives (Intelligence, Engine, Agents, Tools & Memory, Learning) are composed through a declarative *spec* that can be shared, evaluated, and optimized end-to-end (Section 2.1). **(Middle)** Joint accuracy-efficiency evaluation across the evaluated local specs (green; DGX Spark estimates, Appendix D.14) and 3 cloud baselines (blue) reveals that the best on-device configuration (`Qwen3.5-122B`) approaches within 3.2 pp of the best cloud model at up to ∼6,600× lower marginal API cost per query (Section 3.3). **(Right)** LLM-guided spec search uses frontier cloud models as reflective proposers: the teacher diagnoses failures, proposes edits across the full *spec*, and a held-out gate accepts only non-regressing improvements, improving local students by 13–32 pp on average with student training on-device (Section 3.4).

fixed, drops accuracy by 25–39 percentage points across PinchBench and GAIA (Section 3.2, Table 1). This drop reflects both a model-capability gap and a stack-portability problem: retargeting the stack at the same local model recovers 56–77% of it (Table 1). Swapping in a local model forces an all-or-nothing port, requiring that the surrounding stack be re-optimized.

Three challenges prevent building such a local-first personal AI stack:

1. **(C1) Portability is limited by model-specific defaults in personal AI stacks.** Frameworks such as OpenClaw (Steinberger & OpenClaw Contributors, 2025) and Hermes Agent (Nous Research, 2026) expose agent behavior through personal AI stacks whose agentic prompts/reasoning, tool descriptions, memory configuration, and inference runtime settings carry defaults selected for specific cloud models. Effective harness configuration is tightly coupled to the chosen model, so these defaults materially contribute to the substitution gap and make it unrecoverable without retargeting (Section 3.2).

2. **(C2) Accuracy-only evaluation hides the cost of cloud personal AI.** Cloud personal AI creates recurring API spend and gives users no ownership, offline access, or guarantee against price changes. Yet agent benchmarks report accuracy alone, while efficiency tools measure energy or throughput without task success (Mialon et al., 2024; Jimenez et al., 2024; Merrill et al., 2026; You et al., 2023; Hugging Face, 2025; MLCommons, 2024). A local-vs-cloud comparison therefore needs a configuration-level object on which accuracy, latency, energy, power, and marginal API cost are measured together (Section 3.3).

3. **(C3) Shrinking the *frontier-local gap* requires optimizing the full stack.** The hardest reasoning, coding, and research tasks still favor frontier cloud models. We define this gap as the same-evaluation accuracy difference between the best cloud model and the best local configuration. The gap spans model weights, agentic prompts, tool descriptions, memory configuration, and inference runtime choices. Existing optimizers usually edit one component at a time, focusing on improving model weights, agentic logic, or inference runtime configuration (Hu et al., 2022; Khattab et al., 2024; Agrawal et al., 2026; Chen et al., 2024; Ding et al., 2024).

To address these challenges, we present OpenJarvis, an architecture that decomposes any personal AI system into five typed primitives (Section 2.1): (1) *Intelligence*, the language model and its parameters; (2) *Engine*, the inference runtime; (3) *Agents*, the reasoning loop; (4) *Tools & Memory*, data integration and persistent storage; and (5) *Learning*, the optimizer that updates the system from traces. These primitives compose into a *spec*: a typed configuration object on which a single optimization algorithm operates.

We optimize the *spec* with *LLM-guided spec search* (Section 2.3). LLM-guided spec search uses a frontier LM to read and analyze traces generated by the deployed *spec* before proposing edits across the editable primitives. The search only accepts edits that strictly improve the *spec*'s performance, preventing changes addressing one failure mode from causing regressions on another. A single LLM proposal can edit Intelligence weights, Agent prompts, Tool descriptions, Memory configuration, and Engine settings together, moving from anywhere to anywhere else in the complete search space of possible *spec* configurations. The algorithm is deliberately simple: *propose, apply, evaluate, accept*.

In summary, we make three core contributions:

1. **Portable, on-device personal AI (C1).** (Section 3.2). The spec recovers 77% of the `PinchBench` drop and 56–57% of the `GAIA` drop when `Qwen3.5-9B` replaces a cloud model in existing frameworks, while no single retargeted component recovers more than 27% (Table 1). The best on-device spec (`Qwen3.5-122B`) lands within 3.2 pp of the best cloud baseline on average (Table 5).
2. **Joint accuracy-efficiency evaluation (C2).** (Section 3.3). The spec gives local and cloud systems the same configuration-level object of comparison. Under this evaluation, local configurations are Pareto-optimal on marginal API cost and latency, with up to 6,600× lower marginal API cost (`Qwen3.5-122B` running locally) and 4× lower end-to-end latency (`Qwen3.5-35B` running locally) (Figure 3).
3. **LLM-guided spec search (C3).** (Section 3.4). A frontier teacher proposes edits across the spec and a held-out gate accepts only non-regressing improvements. The method improves local students by 13–32 pp on average at 7–11× lower optimization cost than the strongest single-component baseline (LoRA). A proposer-and-move-space ablation shows that both teacher model and the chosen OpenJarvis primitives contribute to performance gains. Across the four benchmarks in this ablation, the LLM proposer beats GEPA-style genetic search by 10.0 pp on average. Furthermore, expanding from the most-edited single primitive to all four for LLM-guided spec search adds 5.5–16.5 pp (Figures 4 and 6).

2 Methods

We present OpenJarvis in three parts: the spec abstraction (Section 2.1), joint accuracy-efficiency evaluation (Section 2.2), and LLM-guided spec search (Section 2.3). The core idea is that every local model needs the surrounding agent system reconfigured around it; the spec makes that reconfiguration explicit and optimizable. Related work is discussed in Appendix A.

2.1 Primitives and the Spec Abstraction

OpenJarvis decomposes a personal AI stack into five primitives, each with a typed interface and registry (Figure 1, left). *Intelligence* specifies the language model and generation parameters. *Engine* specifies the runtime, hardware execution path, batching, quantization, and cache settings. *Agents* specify the reasoning loop, prompts, and tool-use policy. *Tools & Memory* specify external interfaces, retrieval, and persistent user state. *Learning* specifies the optimizer that updates the spec from traces, including weight-only optimizers such as LoRA (Hu et al., 2022), prompt optimizers such as DSPy (Khattab et al., 2024), and LLM-guided spec search (Section 2.3). Implementation details, supported runtimes, connectors, memory systems, and optimizer variants are in Appendix B.1.

Five-primitive architecture. Each primitive is an independent degree of freedom that prior systems optimize in isolation. The decomposition is designed to separate choices that are independently configurable in deployed systems. For example, Intelligence and Engine separate model selection (`Qwen3.5-9B` vs. `Qwen3.5-122B`) from runtime selection (`vLLM` vs. `Ollama`). Agents and Tools separate the reasoning loop (ReAct vs. CodeAct) from the interfaces it invokes (filesystem, web search, MCP servers). Learning is structurally different from the other four primitives: it is the primitive we use to optimize all the others, making edits to Intelligence weights, Agent prompts/logic, Tools & Memory selection and descriptions, and Engine runtime specifications. This primitive lets the same framework describe systems that share Intelligence and Agent configurations but differ only in Learning policy, such as OpenClaw-RL (Wang et al., 2026b). Section 3.4 tests whether these distinctions between primitives matter for constructing and optimizing local AI.

(a) The spec: a typed configuration object

```
Spec S := {
    Intelligence:  (model, params, quant),
    Engine  :  (backend, batch, kv_cache),
    Agent   :  (loop, prompts,
                tool_strategy),
    Tools   :  (set, descriptions, memory),
    Learning :  (optimizer, reward, gate)
}
```

Each slot is independently swappable; individual edits target a slot; batches can span slots.

(b) A single optimization algorithm operates on the spec

```
Optimize(S, T, G, ϵ, k, B) → S*:
  while stagnant < k and cost < B:
    failures = Teacher.diagnose(traces(S))
    edits    = Teacher.plan(S, failures)
    S        = accept(S, edits, Gate, ϵ)
  return S
```

Edits can span editable primitives; the optimizer design space (LoRA, DSPy, GEPA, LLM-guided spec search) is parameterized by which edit types are permitted.

Figure 2: **The spec abstraction.** **(a)** A spec is a typed configuration object with 5 primitives: Intelligence, Engine, Agents, Tools & Memory, and Learning. **(b)** Optimizers instantiate the same signature by restricting which fields they edit. LoRA edits Intelligence weights; DSPy and GEPA, as originally published, edit Agent prompts; LLM-guided spec search edits Intelligence, Engine, Agents, and Tools & Memory jointly.

The spec abstraction. The five primitives are composed through the *spec*: a typed configuration object on which a single optimization algorithm operates. Figure 2 shows the spec as a structured declaration alongside the optimization signature; Figure 8 (Appendix B.1) shows the corresponding TOML serialization. A spec can be versioned, shared, evaluated on standardized benchmarks, and optimized end-to-end (Section 2.3).

2.2 Evaluation Metrics

We evaluate complete specs rather than isolated models. For each spec, an instrumented wrapper records five quantities per query: accuracy, energy, latency, power, and dollar cost. Accuracy is exact match or LLM-judge win rate, depending on the benchmark. For open-weight models, the wrapper measures energy on the local hardware (the local values plotted in Figures 1 and 3 are DGX Spark estimates; Appendix D.14); closed cloud models cannot be run locally, so we do not report their energy. Latency is end-to-end wall-clock time, and power is energy divided by latency. Dollar cost is per-token API pricing and tool fees; local model inference is reported as $0 marginal API cost, with hardware and electricity reported separately (total cost of ownership in Appendix D.10). Together, the spec and wrapper expose Pareto structure that accuracy-only benchmarks and single-axis efficiency tools miss (Mialon et al., 2024; Jimenez et al., 2024; Merrill et al., 2026; You et al., 2023; Hugging Face, 2025; MLCommons, 2024).

2.3 LLM-Guided Spec Search

We optimize configurations by searching over specs. A *spec* is one complete local AI configuration; changing its primitive fields defines the search space. The Learning primitive specifies how a spec is updated from traces: which edits are considered, how edited specs are evaluated, and when optimization stops. In *LLM-guided spec search*, a frontier model reads eligible traces, identifies failure patterns, and proposes candidate edits. For this search, the frontier model identifies *failure cluster*, groups of traces sharing a common failure mode. These are annotated with student vs. teacher success rates and a natural-language characterization of the skill gap (e.g., "student fails on multi-hop questions requiring calendar lookups because it does not invoke the calendar tool"); see Appendix B.4 for the full clustering protocol. For the search, we evaluate each edited spec on held-out examples for the task, keeping the edit only if it improves the targeted failures without causing unacceptable regressions elsewhere. We refer to this held-out validation check as the gate. Intelligence edits change model parameters or training triggers. Engine edits change runtime and serving choices. Agent edits change prompts, reasoning loops, and verification. Tools & Memory edits change Tool availability, Tool descriptions, and Memory configuration. Cloud-as-tool use is disabled in the headline local configurations. At inference time, the resulting spec runs on-device for model inference and agent execution. When an Intelligence edit triggers student training, that training also runs locally. Teacher calls provide diagnoses, edit proposals, and labels, not inference-time model calls.

GEPA (Agrawal et al., 2026) is the closest prior search method: it also uses LLM-based reflection over trajectories, but couples reflection to population, merge, and Pareto-frontier machinery. Our contrast is therefore not "LLM proposer versus no LLM proposer." The contrast is representation and acceptance. GEPA-style search maintains a population of candidate specs and merges high-performing candidates. OPENJARVIS uses a simpler loop: propose one edit, evaluate the edited spec on held-out examples, and keep it only if performance improves without unacceptable regressions. The key difference is that an OPENJARVIS edit can change any of the four editable primitives in the same spec. The defining property of the spec is that Intelligence, Engine, Agent, and Tools & Memory fields are degrees of freedom in one optimizable object. Figure 9 (Appendix B.4) summarizes the three regimes compared in Section 3.4: LLM-guided spec search (Algorithm 1), GEPA-style search (Algorithm 2), and single-component optimization (Algorithm 3). Figure 6 separates the proposer axis from the move-space axis.

Search loop. Each session repeats four steps: diagnose failures, propose edits, execute candidates, and commit only gated improvements. Diagnose clusters traces where the student underperforms; on `LiveCodeBench`, for example, the teacher traced 31 of 50 `Qwen3.5-9B` failures to unexecuted code. Plan proposes edits over Intelligence, Engine, Agent, and Tool fields; here, a run-and-check Agent prompt and a stricter tool contract. Execute evaluates each edited spec on a held-out gate built from eligible traces, agentic datasets with known answers (RJT1990, 2025; Su et al., 2026), and standard benchmark splits (Wang et al., 2024b; Mialon et al., 2024; Yao et al., 2025); the two edits above gained 11.7 and 6.2 pp. Repeat commits accepted edits and rolls back rejected ones. The loop stops when gate scores stagnate for k sessions (default k=5) or the budget is exhausted. Appendix B.4 walks through each phase, and Appendix D.7 gives the full before-and-after specs.

Gate score. Let $G(S)$ be the held-out gate score of spec S, and let $G_c(S)$ be the score restricted to failure cluster c. For a feasible edit e targeting cluster c, with $S' = \mathrm{apply}(S, e)$, we accept if and only if

$$\big[G_c(S') > G_c(S) \text{ or } (c \in \mathcal{C}_{\text{eff}} \wedge J_c(S') < J_c(S))\big] \quad \text{and} \quad G_{c'}(S') \geq G_{c'}(S) - \epsilon \quad \forall c' \neq c, \tag{1}$$

where $\mathcal{C}_{\text{eff}}$ are efficiency-bounded clusters and $J_c = w_E\hat{E} + w_L\hat{L} + w_C\hat{C}$ is a user-weighted efficiency score on c's traces; proposals that cannot load, exceed 90% of usable memory, or miss the tier's p95 latency target are rejected before evaluation. The default tolerance is $\epsilon = 1\%$. This single rule applies across Intelligence weights, Agent prompts, Tool descriptions, Memory configuration, and Engine/runtime settings, so energy, latency, and cost are optimized for non-training edits after feasibility filtering, not only enforced as constraints.

Training-internal reward for Intelligence edits. If Intelligence edits are requested involving model training, each candidate response y to a query q is scored by a composite reward over accuracy, energy, latency, and cost:

$$R(q, y) = \alpha R_{\text{acc}}(q, y) - \beta\hat{E}(q, y) - \gamma\hat{L}(q, y) - \delta\hat{C}(q, y). \tag{2}$$

The default weights are $(\alpha, \beta, \gamma, \delta) = (0.5, 0.1, 0.1, 0.3)$. The reward scores responses during GRPO training within an Intelligence edit. The reward is internal to training; the gate (Equation 1) evaluates the resulting spec as a whole. Normalization details and robustness checks are in Appendix D.5.

3 EXPERIMENTS

3.1 MODELS, BENCHMARKS, AND HARDWARE

We evaluate OPENJARVIS across 8 benchmarks, 11 local models from 4 families, and 3 cloud baselines; Appendix C.4 lists the 7 hardware platforms. The suite contains 508 tasks, and agentic benchmarks enforce a 2-hour timeout per task. All results report the mean over 5 independent runs. Scoring uses `GPT-5-mini` as judge except where benchmarks provide deterministic grading. We define personal compute as a single-user, desk-side or portable system administered by the user, with inference and data local; our platforms span roughly $600–9,000. Appendix C details benchmarks (Appendix C.1), model choices and full accuracy tables (Appendices C.2 and C.3), hardware (Appendix C.4), and protocol (Appendix C.5).

Table 1: **Substitution and per-component retargeting at fixed Intelligence.** Frameworks: OpenClaw (Steinberger & OpenClaw Contributors, 2025) and Hermes Agent (Nous Research, 2026). *Cloud*: default framework with `Claude Opus 4.6`; *Qwen*: default framework with `Qwen3.5-9B`. The next five columns retarget one component of the OPENJARVIS spec (or Agent and Tools together, *A+T*) with Intelligence fixed at `Qwen3.5-9B`, holding weights, items, seeds, tools, device, judge, and decoding identical; *Full* retargets Engine, Agent, and Tools & Memory. *Recovery* = (Full − Qwen)/(Cloud − Qwen). Accuracy (%) is the mean of 5 runs (standard deviation 1.2–2.2).

				Retarget one component						
Framework	**Task**	**Cloud**	**Qwen**	Agent	Tools	Memory	Engine	A+T	**Full**	**Recovery**
OpenClaw	`PinchBench`	96.0	62.3	70.8	68.7	64.9	66.0	81.2	88.4	77.4%
	`GAIA`	58.0	19.2	27.6	25.4	21.8	23.3	34.9	41.5	57.5%
Hermes Agent	`PinchBench`	93.5	68.7	75.4	72.8	70.2	71.0	82.0	87.9	77.4%
	`GAIA`	55.1	22.4	29.8	27.0	24.0	25.1	35.6	40.8	56.3%

3.2 THE SPEC RECOVERS 56–77% OF THE DROP AND LANDS WITHIN 3.2 PP OF CLOUD ON AVERAGE (C1)

Does the spec abstraction (Section 2.1) restore portability when local models are substituted into cloud-designed frameworks? We test this with a controlled portability experiment on two production frameworks (Table 1) and summarize the local-vs-cloud accuracy frontier using the full accuracy table in Appendix C.3.

OPENJARVIS reduces the local-substitution drop from 25–39 pp to 5.6–16.5 pp. Retargeting the spec around the same local model recovers most of the substitution drop (Table 1). Replacing `Claude Opus 4.6` with `Qwen3.5-9B` in each framework drops accuracy by 24.8–38.8 pp. With the same local model under an OPENJARVIS spec, the remaining drop relative to the cloud framework is only 5.6–16.5 pp. Equivalently, the spec closes 77% of the `PinchBench` drop and 56–57% of the `GAIA` drop. This bounds the contribution of model-specific defaults; the residual 5.6–16.5 pp gap is capability-bound, consistent with `GAIA`'s deep-reasoning demands exceeding a 9B model regardless of configuration. OpenClaw and Hermes Agent package Agent prompts, Tool descriptions, Memory configuration, Engine/runtime settings, and model-specific output expectations into personal AI stacks tuned to their intended cloud models. The OPENJARVIS *spec* exposes these assumptions as typed slots. Engine, Agent, and Tool fields can then be retargeted while holding Intelligence fixed. Every single-component retarget has a positive point estimate, but none closes more than 27.0% of the drop, and Agent and Tools interact positively (paired-bootstrap 95% CIs from $[+0.2, +2.0]$ to $[+2.1, +5.8]$). Across all eight benchmarks, substitution lowers accuracy in all 16 framework–benchmark comparisons and retargeting recovers 56.3–77.4% of each loss (Appendix C.6, with the responsible defaults in source).

`Qwen3.5-122B` lands within 3.2 pp of the best cloud model on average, and the per-benchmark best local model matches or exceeds cloud on 4 of 8 benchmarks. Across the evaluated local specs, the best single on-device spec, `Qwen3.5-122B`, reaches 80.3% average accuracy, within 3.2 pp of `Claude Opus 4.6` at 83.5% (Appendix C.3, Table 5). Taking the best local model per benchmark (the *Best Local* row of Table 5), local models match or exceed the best cloud score on 4 of 8 benchmarks: `ToolCall-15` (`Nemotron-Super-120B`), `PinchBench` (`Qwen3.5-27B`, `-35B`, and `-122B`), `LiveCodeBench` (`Gemma4-26B`), and τ-Bench V2 (`Qwen3.5-122B`). The remaining gaps concentrate on `GAIA`, τ^2-Bench Telecom, `DeepResearchBench`, and `LiveResearchBench`, motivating the search experiments in Section 3.4.

3.3 LOCAL MODELS TRADE 3.2 PP ACCURACY FOR UP TO 6,600× LOWER COST AND 4× LOWER LATENCY (C2)

Heavy users of cloud personal AI spend thousands per year on one assistant, and none of it buys ownership, offline access, or price stability. Can on-device specs recover most of this cost while

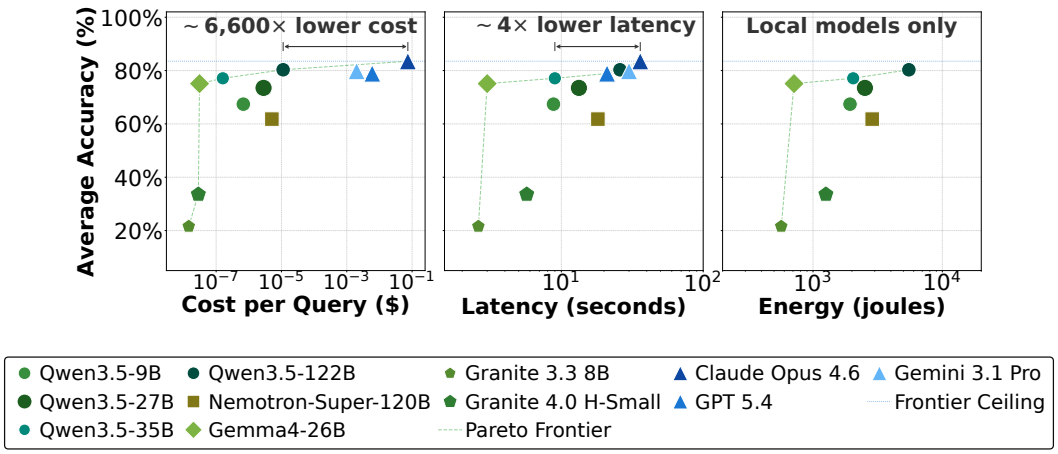


Figure 3: **Accuracy-efficiency frontier.** `Qwen3.5-122B` approaches the best cloud accuracy within 3.2 pp while reducing marginal API cost by up to ∼6,600×, and `Qwen3.5-35B` reduces end-to-end latency by roughly 4× under our benchmark protocol. Local latency and energy are estimates for a DGX Spark; the energy panel shows local models only. Model, quantization, and platform for every point are in Appendix D.14.

preserving accuracy? Treating the spec as the unit of comparison and instrumenting every query (Section 2.2), we compare local and cloud directly.

Local configurations define the accuracy and cost frontier. Apart from the most accurate cloud model (and `GPT 5.4` on latency), every non-dominated configuration on cost and latency in Figure 3 is local: local inference replaces the per-token API charge with a tool fee of at most about a thousandth of a cent per query (`Qwen3.5-122B`; mostly free-tier or cached search calls). `Qwen3.5-122B` trades 3.2 pp of accuracy (80.3% vs. 83.5%) for up to a ∼6,600× lower marginal API cost than `Claude Opus 4.6` at $0.0742 per query with all input billed at the full rate; with prompt caching (at least $0.0329 per query) and local electricity counted, the advantage is ∼120× (Appendix D.10). The cost axis reports marginal API fees, not total cost of ownership; counting acquisition but not electricity, break-even against the cached lower bound ranges from 18,200 queries ($599 RX 9070 XT) to about 270,500 ($8,900 RTX 6000 Pro; accelerator-only prices). `Qwen3.5-35B` also completes the full agentic workloads roughly 4× faster than `Claude Opus 4.6` in our protocol, though single-shot prompts can favor cloud serving due to time-to-first-token optimizations. Under device-feasibility constraints, spec search reaches 80.7% on a 24 GB Mac Mini (`Qwen3.5-9B`, Q4) and 84.1% on a 128 GB DGX Spark (`Qwen3.5-122B`, FP4; Appendix D.8).

3.4 LLM-Guided Spec Search Improves Local Students by 13–32 pp at 7–11× Lower Cost (C3)

Sections 3.2 and 3.3 show that smaller students such as `Qwen3.5-9B` still trail cloud; we evaluate whether LLM-guided spec search closes this gap on `PinchBench`, `LiveCodeBench`, and `LiveResearchBench`. For each benchmark, we run search with four local target models and three frontier proposer models; Appendix D.3 extends the evaluation to all 8 benchmarks.

LLM-guided spec search improves OpenJarvis students on personal AI benchmarks. Search improves every reported student–teacher pair on all three benchmarks (Figure 4), and the gains come from joint edits across Intelligence, Agent, Tool, Memory, and Engine primitives rather than any single component (Figure 6). Across the eight-benchmark suite, average gains per student model range from 13.1 pp (`Gemma4-E4B`) to 31.5 pp (`Nemotron-Nano-4B`; Table 8).

LLM-guided spec search improves accuracy at lower optimization cost. Spec search is cheaper to optimize than weight training because of the acceptance gate: rejected edits never trigger full

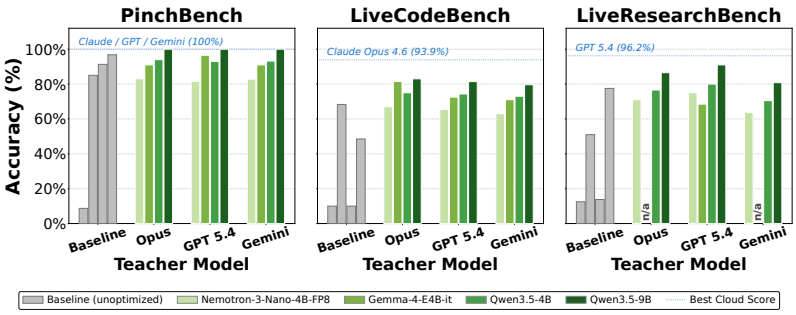


Figure 4: **LLM-guided spec search improves local specs.** Every reported student–teacher pair improves over the unoptimized spec (Table 8) on all three benchmarks; *n/a* marks two `Gemma4-E4B` cells not reported. The strongest search-optimized `Qwen3.5-9B` student reaches 100.0% on `PinchBench`, 83.0% on `LiveCodeBench`, and 91.0% on `LiveResearchBench`.

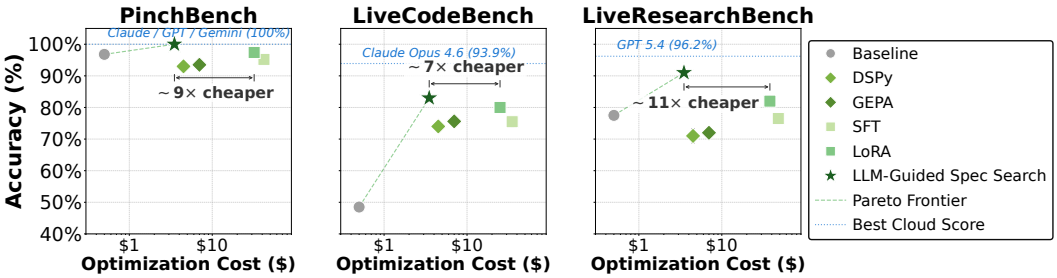


Figure 5: **Accuracy vs. optimization cost.** LLM-guided spec search reaches the best accuracy on all three main benchmarks. LoRA is the strongest single-component baseline, but LLM-guided spec search is 7.1–10.9× cheaper to optimize. Spec-search cost is for one search with the best teacher ($3.5); searching with all three teachers costs $15.60 per task (Appendix D.2).

training runs, and accepted edits compound across sessions. With `Qwen3.5-9B` as the student and under the same offline protocol as DSPy/SIMBA, prompt-only GEPA, SFT, and LoRA (Khattab et al., 2024; Agrawal et al., 2026; Hu et al., 2022), it reaches the best accuracy on all three benchmarks and beats LoRA, the strongest single-component baseline, by 2.6–9.0 pp at 7.1–10.9× lower optimization cost (Figure 5). Matched tuning helps the cloud baseline far less: the same search raises `Claude Opus 4.6` from 83.5% to 84.6%, versus +14.9 pp for `Qwen3.5-9B` (Appendix D.9).

Both proposer and move space drive the gain. The gain needs both the LLM proposer and the full four-primitive move space; the edit catalog and gate alone do not explain it (Figure 6). For `Qwen3.5-9B` with a `Claude Opus 4.6` teacher and all four primitives editable, the LLM proposer beats GEPA-style genetic search by 10.0 pp and random sampling from the edit catalog by 14.0 pp on average. With the LLM proposer fixed, restricting edits to the most-edited single primitive costs 5.5–16.5 pp of accuracy and the 2.65–3.45× latency speedup, and merging primitive pairs that deployed systems treat as independent lowers accuracy by 2.8–9.7 pp and the speedup to 1.47–1.62× (Table 13; baselines in Appendix D.6).

What the optimizer learns. The teacher largely routes each failure type to a matching primitive. By benchmark, the dominant primitive matches task structure: Intelligence edits dominate code (44% on LiveCodeBench), Agent edits dominate customer-service and agentic tasks (41–45% on PB, TauB, and TBTel), and Tool edits dominate tool-calling and research (39–47% on TC15, GAIA, DRB, and LRB; Appendix D.4, Figure 10). By failure category, the teacher maps each diagnosis to the corresponding primitive: retrieval failures receive mostly Tool edits, reasoning failures mostly Intelligence edits, control-flow failures mostly Agent edits, and efficiency-bounded failures mostly Engine edits (Figure 11). Intelligence edits make up 16–44% of accepted edits across benchmarks, and only 8.0% of accepted edits in the `Qwen3.5-9B` searches update weights (adapter-only QLoRA

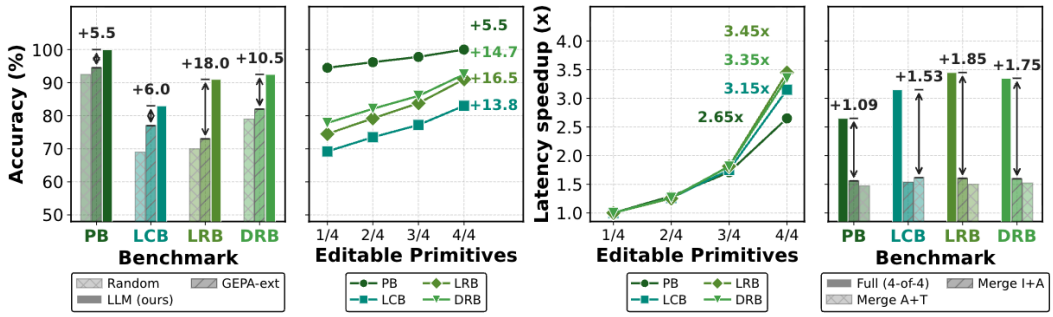


Figure 6: **Proposer and move-space ablations for LLM-guided spec search.** Student: `Qwen3.5-9B`; teacher: `Claude Opus 4.6`. Panels from left to right: **(a)** at fixed four-primitive move space, the LLM proposer outperforms a template-random proposer and GEPA-style genetic search that edits all four primitives under a matched metric-call budget; **(b, c)** at fixed LLM proposer, expanding from the most-edited single primitive to all four improves both accuracy and latency; **(d)** merging primitive pairs reduces speedup relative to the full four-primitive spec. The 4-of-4 values match the best-teacher results of Table 8, whose `LiveResearchBench` spec comes from `GPT 5.4`.

or GRPO); the remainder span Engine, Agent, and Tool fields (Table 9). This explains why no single primitive reaches the joint-search ceiling: the useful intervention depends on the failure mode, and many failures require coordinated changes across multiple primitives. Weight training is therefore a targeted escalation rather than the main lever: without any weight updates the `Qwen3.5-9B` searches still gain 12.0 of their 14.9 pp (79.4% vs. 82.3%, from 67.4%), and each adapter update is a one-time scheduled cost with serving paused rather than a per-query load (Appendix D.12). Every proposal is logged with its diagnosis, gate delta, and decision (Appendix D.7).

4 Discussion and Conclusion

OpenJarvis shows that local personal AI requires more than replacing a cloud model with an open-weight model. The surrounding stack must be portable, measurable, and optimizable around the local model. The spec abstraction provides the needed representation: it makes model substitution recoverable (C1), exposes the local-cloud accuracy-efficiency frontier (C2), and gives LLM-guided spec search a target for transferring frontier-model guidance into an on-device system (C3).

Frontier proposer models are search-time only. A natural concern is that LLM-guided spec search reintroduces cloud dependence. The dependency is bounded: frontier models are called during search and zero times for inference-time model execution. The best undistilled local model per benchmark already matches or beats the best cloud score on 4 of 8 benchmarks, so frontier proposer models are not required for local systems to function. When a frontier proposer is used, only eligible traces are transmitted according to the protocol in Appendix B.3. At 100 queries per day, the amortized proposer cost falls below $0.001 per query within six months (Appendix D.2). Search is task-specific, but specs transfer to related tasks with 79–83% of the target-specific gain, and one pooled search over the personal workload reaches 79.8% for $41.20, versus $124.80 for eight task-specific searches (Appendix D.13).

Robustness and limitations. Reward-weight variants, search-seed variance, and random-restart gains are all smaller than the 13–32 pp headline gain (Appendix D.5). Cloud APIs sit at ceiling on several benchmarks and wrap models in harnesses we cannot reach. Remaining limitations include statistical precision from 5 runs per configuration, possible judge bias from `GPT-5-mini`, and single-machine evaluation only (Appendix E). Broader impacts and release safeguards are discussed in Appendix F.

AI USE STATEMENT

We used generative AI tools to aid and polish the writing of this manuscript, including citation verification and checking the consistency of reported results across sections. Separately, LLMs are components of the method we study: frontier models act as teachers that generate synthetic traces and labels and propose spec edits, and `GPT-5-mini` serves as an evaluation judge (Sections 2.3 and 3.1, Appendix B). Beyond these, we did not use generative AI tools for any task for which the ICLR 2027 AI Policy for Authors requires disclosure. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies personal AI systems that operate over sensitive user data, such as email, messages, calendars, and health records, through the Tools & Memory primitive. In the reference implementation, ordinary inference keeps model inference, Engine execution, Agent state, Memory contents, and telemetry on-device; each Connector requires an OAuth consent flow, and cross-Connector queries are audit-logged (Appendices B.2 and F). When a cloud teacher is enabled for LLM-guided spec search, only user-eligible traces that have passed a sensitive-data scrubber are transmitted, and only during the bounded search phase (Appendix B.3). Because the same architecture could be repurposed for surveillance or used to automate harmful tasks, the framework places safeguards at the Tools boundary (sensitive-data scanning, prompt-injection detection, SSRF protection, and sandboxed execution); these safeguards are not foolproof, and we discuss the remaining risks in Appendix F. The terms of service of frontier-model providers restrict some uses of model outputs, so users of LLM-guided spec search should verify compliance with their teacher provider's current terms before initiating search (Appendix F). This research involves no human subjects; all evaluations use publicly available benchmarks, open-weight models under their original licenses (Table 4), and commercial model APIs.

REPRODUCIBILITY STATEMENT

Section 3.1 and Appendix C describe the benchmark suite (Appendix C.1, Table 3), the evaluated models with their quantization formats and serving engines (Appendix C.2, Table 4), the hardware platforms (Appendix C.4, Table 6), and the evaluation protocol (Appendix C.5). Section 2.3 and Appendix B.4 specify LLM-guided spec search, including the gate rule, regression tolerance, stopping criterion, and composite-reward weights; Appendix D.6 specifies the proposer-ablation baselines, and Appendix D.5 reports robustness to search seeds, reward weights, and random restarts. The code, configurations, and experiment scripts are included, anonymized, in the supplementary material of this submission. The complete code, data, and experimental scripts will also be released as a .zip archive with the camera-ready version.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, pp. 8479–8565, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r. arXiv:2507.19457.

Anthropic. Model Context Protocol. https://modelcontextprotocol.io, 2024.

Anthropic. Pricing. Claude Platform documentation, https://platform.claude.com/docs/en/about-claude/pricing, 2026a.

Anthropic. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026b. Released February 5, 2026.

Apple Inc. Apple introduces M4 chip. Apple Newsroom, https://www.apple.com/newsroom/2024/05/apple-introduces-m4-chip/, 2024. Published May 7, 2024.

Parth Asawa, Alan Zhu, Abigail O'Neill, Matei Zaharia, Alexandros G. Dimakis, and Joseph E. Gonzalez. How to train your advisor: Steering black-box LLMs with advisor models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2510.02453.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2506.07982.

Harrison Chase. LangChain. https://github.com/langchain-ai/langchain, 2022.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. arXiv:2305.05176.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 8359–8388, 2024.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 758–759. ACM, 2009. doi: 10.1145/1571941.1572114.

CrewAI, Inc. CrewAI: Framework for orchestrating role-playing, autonomous AI agents. https://github.com/crewAIInc/crewAI, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115, 2023.

Ettore Di Giacinto. LocalAI: The free, open source OpenAI alternative. https://github.com/mudler/LocalAI, 2023.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2024. arXiv:2404.14618.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Licheng Zhang, Xiaorui Wang, and Zhendong Mao. DeepResearch Bench: A comprehensive benchmark for deep research agents. In *The Fourteenth International Conference on Learning Representations*, pp. 42414–42448, 2026. URL https://openreview.net/forum?id=hQ0K2Hhq7H. arXiv:2506.11763.

Element Labs. LM Studio. https://lmstudio.ai/docs/app, 2023.

Clement Farabet and Olivier Lacombe. Gemma 4: Byte for byte, the most capable open models. Google blog (The Keyword), https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/, 2026. Published April 2, 2026.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, Ge Liu, and Jiaxuan You. LLMRouter: Unified infrastructure for developing, evaluating, and deploying LLM routers. *arXiv preprint arXiv:2608.06867*, 2026. URL https://arxiv.org/abs/2608.06867.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. Also known as GPTQ; arXiv:2210.17323.

Yonggan Fu, Xin Dong, Shizhe Diao, Matthijs Van keirsbilck, Hanrong Ye, Wonmin Byeon, Yashaswi Karnati, Lucas Liebenwein, Maksim Khadkevich, Alexander Keller, Jan Kautz, Yingyan Celine Lin, and Pavlo Molchanov. Nemotron-Flash: Towards latency-optimal hybrid small language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5388. URL https://papers.nips.cc/paper_files/paper/2025/hash/ec2691f6b5c88eb650554dffbc0e0af5-Abstract-Conference.html. arXiv:2511.18890.

Gartner, Inc. Gartner predicts that by 2030, performing inference on an LLM with 1 trillion parameters will cost GenAI providers over 90% less than in 2025. Gartner Press Release, https://www.gartner.com/en/newsroom/press-releases/2026-03-25-gartner-predicts-that-by-2030-performing-inference-on-an-llm-with-1-trillion-parameters-will-cost-genai-providers-over-90-percent-less-than-in-2025, March 2026.

Gemma Team. Gemma 3n. https://ai.google.dev/gemma/docs/gemma-3n, 2025.

Jiaxiang Geng, Lunyu Zhao, Yiyi Lu, and Bing Luo. MobileFineTuner: A mobile-native framework for on-device LLM fine-tuning in real-world embedded AI applications. *arXiv preprint arXiv:2512.08211*, 2025.

Georgi Gerganov. llama.cpp. https://github.com/ggml-org/llama.cpp, 2023.

Google. gemma.cpp. https://github.com/google/gemma.cpp, 2024.

Google. Agent Development Kit (ADK). https://github.com/google/adk-python, 2025.

Google. Gemini Nano. https://developer.android.com/ai/gemini-nano, 2026. Android Developers documentation.

Google DeepMind. Gemini 3.1 Pro. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026. Model card; released February 19, 2026.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, et al. Apple Intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. arXiv:2501.12948.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

HKUDS. nanobot. https://github.com/HKUDS/nanobot, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations*, 2022.

Hugging Face. AI Energy Score: Initiative to establish comparable energy efficiency ratings for AI models. https://huggingface.github.io/AIEnergyScore/, 2025.

IBM Granite Team. Granite 4.0 language models. https://github.com/ibm-granite/granite-4.0-language-models, 2025a. Models released October 2, 2025.

IBM Granite Team. Granite-3.3-8B-Instruct. https://huggingface.co/ibm-granite/granite-3.3-8b-instruct, 2025b. Model card; released April 16, 2025.

Intel Corporation. Intel Core Ultra processors (Series 1). https://www.intel.com/content/www/us/en/ark/products/series/236803/intel-core-ultra-processors-series-1.html, 2023. Code name Meteor Lake; launched Q4 2023.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, pp. 58791–58831, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, pp. 54928–54958, 2024. arXiv:2310.03714.

Kilo Code. PinchBench: Benchmarking LLM models as OpenClaw coding agents. https://github.com/pinchbench/skill, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Steven Lei. ToolCall-15. https://github.com/stevibe/ToolCall-15, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286, 2023.

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256KB memory. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22941–22954, 2022.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024.

Liquid AI. Introducing Liquid Nanos: Frontier-grade performance on everyday devices. https://www.liquid.ai/blog/introducing-liquid-nanos-frontier-grade-performance-on-everyday-devices, September 2025.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32431–32454, 2024.

Aisha Malik. ChatGPT reaches 900M weekly active users. TechCrunch, https://techcrunch.com/2026/02/27/chatgpt-reaches-900m-weekly-active-users/, February 2026.

MemoV. MimiClaw: Pocket AI assistant on a $5 chip. https://github.com/memovai/mimiclaw, 2026.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/25ae35b5b1738d80f1f03a8713e405ec-Abstract-Conference.html.

MLC team. MLC-LLM: Universal LLM deployment engine with ML compilation. https://github.com/mlc-ai/mlc-llm, 2023.

MLCommons. Reference implementations of MLPerf inference benchmarks. https://github.com/mlcommons/inference, 2024.

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala. Ludwig: A type-based declarative deep learning toolbox. *arXiv preprint arXiv:1909.07930*, 2019.

Mergen Nachin, Digant Desai, Stephen Jia, Chen Lai, Mengwei Liu, Jacob Szwejbka, Raziel Alvarez, RJ Ascani, Dave Bort, Manuel Candales, et al. ExecuTorch – a unified PyTorch solution to run ML models on-device. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 569–609, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/236f915dd02af4f11927f67330b21d4b-Abstract-Conference.html. arXiv:2605.08195.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Ré. Cost-efficient collaboration between on-device and cloud language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 45682–45719, 2025. arXiv:2502.15964.

National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology, U.S. Department of Commerce, January 2023. URL https://doi.org/10.6028/NIST.AI.100-1.

NEAR AI. IronClaw. https://github.com/nearai/ironclaw, 2026.

Nous Research. Hermes Agent: The agent that grows with you. https://github.com/NousResearch/hermes-agent, 2026.

NVIDIA. NVIDIA Nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

Ollama. Ollama. https://github.com/ollama/ollama, 2023.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026a. Released March 5, 2026.

OpenAI. An open-source spec for Codex orchestration: Symphony. https://openai.com/index/open-source-codex-orchestration-symphony/, April 2026b. Specification and reference implementation: https://github.com/openai/symphony.

OpenBMB. EdgeClaw: Edge-cloud collaborative personal AI assistant based on OpenClaw. https://github.com/OpenBMB/EdgeClaw, 2026.

OWASP Foundation. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Konstantin F. Pilz, Yusuf Mahmood, and Lennart Heim. AI's power requirements under exponential growth: Extrapolating AI data center power demand and assessing its potential impact on U.S. competitiveness. Research Report RR-A3572-1, RAND Corporation, 2025. URL https://www.rand.org/pubs/research_reports/RRA3572-1.html.

Qualcomm Technologies. Qualcomm Hexagon NPU. https://www.qualcomm.com/processors/hexagon, 2025.

Qwen Team. Qwen-Agent. https://github.com/QwenLM/Qwen-Agent, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026.

Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. Overton: A data system for monitoring and improving machine-learned products. In *10th Conference on Innovative Data Systems Research (CIDR)*, 2020. URL https://www.cidrdb.org/cidr2020/papers/p33-re-cidr20.pdf. arXiv:1909.05372.

RJT1990. GeneralThought-430K. https://huggingface.co/datasets/RJT1990/GeneralThoughtArchive, 2025.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, number 500-225 in NIST Special Publication, pp. 109–126. NIST, 1994.

Jon Saad-Falcon, Avanika Narayan, Hakki Orhun Akengin, J. Wes Griffin, Herumb Shandilya, Adrian Gamarra Lafuente, Medhya Goel, Rebecca Joseph, Shlok Natarajan, Etash Kumar Guha, Shang Zhu, Ben Athiwaratkun, John Hennessy, Azalia Mirhoseini, and Christopher Ré. Intelligence per watt: Measuring intelligence efficiency of local AI. *arXiv preprint arXiv:2511.07885*, 2025.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3715–3734, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.272. URL https://aclanthology.org/2022.naacl-main.272/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Sipeed. PicoClaw: Ultra-efficient AI assistant in Go. https://github.com/sipeed/picoclaw, 2026.

Subash SN, Akshay Nambiar, Patrik Lambert, Milan Gritta, Gianfranco Cordella, and Amril Nurman. An edge-first generalized LLM LoRA fine-tuning framework for heterogeneous GPUs. Hugging Face Blog (QVAC by Tether), https://huggingface.co/blog/qvac/fabric-llm-finetune, 2025. Published December 1, 2025.

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-tuning and prompt optimization: Two great steps that work better together. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10696–10710. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.597.

Peter Steinberger and OpenClaw Contributors. OpenClaw: Personal AI assistant. https://github.com/openclaw/openclaw, 2025. Formerly Clawdbot/Moltbot; renamed OpenClaw in January 2026.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating intelligence via efficient model and tool orchestration. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2511.21689.

U.S. Energy Information Administration. Electric Power Monthly, Table 5.3: Average price of electricity to ultimate customers: Total by end-use sector. https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_3, 2026. Data through June 2026; released August 26, 2026. Archived at https://web.archive.org/web/20260904171311/https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=epmt_5_3.

Jiayu Wang, Yifei Ming, Riya Dulepet, Qinglin Chen, Austin Xu, Zixuan Ke, Frederic Sala, Aws Albarghouthi, Caiming Xiong, and Shafiq Joty. LiveResearchBench: A live benchmark for user-centric deep research in the wild. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024a. arXiv:2402.01030.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. OpenClaw-RL: Train any agent simply by talking. *arXiv preprint arXiv:2603.10165*, 2026b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 95266–95290, 2024b.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. arXiv:2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, pp. 9965–10017, 2025.

Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 119–139, 2023. Software: https://ml.energy/zeus/.

ZeroClaw Labs. ZeroClaw: Personal AI assistant. https://github.com/zeroclaw-labs/zeroclaw, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026. arXiv:2510.04618.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

A Related Work

Unifying abstractions have historically accelerated ML progress. Imperative deep-learning frameworks (Paszke et al., 2019) and type-based declarative ML toolboxes (Molino et al., 2019) unified model construction, while data systems for monitoring deployed ML products (Ré et al., 2020) and LM pipeline compilers (Khattab et al., 2024) unified iterative improvement. The personal AI ecosystem is undergoing a similar fragmentation-to-unification transition. Existing tools each formalize a subset of the five primitives (Intelligence, Engine, Agents, Tools & Memory, and Learning) while hardcoding or omitting the rest.

Agents and Tools: personal AI frameworks. Personal AI frameworks, including OpenClaw (Steinberger & OpenClaw Contributors, 2025), Hermes Agent (Nous Research, 2026), LangChain (Chase, 2022), CrewAI (CrewAI, Inc., 2024), Google ADK (Google, 2025), OpenAI Symphony (OpenAI, 2026b), Qwen-Agent (Qwen Team, 2023), and related systems (Sipeed, 2026; OpenBMB, 2026; HKUDS, 2026; NEAR AI, 2026; ZeroClaw Labs, 2026; MemoV, 2026), formalize Agents and Tools but often package them as personal AI stacks tuned to specific cloud models. Their Agent prompts, Tool descriptions, Memory configuration, and Engine/runtime settings are tied to specific model and engine choices. When a different Intelligence is substituted, these model-specific defaults materially contribute to the resulting accuracy drop (Section 3.2). Closed systems (Apple Intelligence (Gunter et al., 2024), Gemini Nano (Google, 2026)) additionally hardcode the Engine to proprietary runtimes and expose no Learning interface. In each case, the Agents and Tools layers exist as configurable layers; Intelligence, Engine, and Learning do not.

Engine and Intelligence: local inference tools. Local inference tools (Ollama (Ollama, 2023), llama.cpp (Gerganov, 2023), LM Studio (Element Labs, 2023), LocalAI (Di Giacinto, 2023), gemma.cpp (Google, 2024)) formalize Engine as a configurable layer and expose Intelligence as a configurable model slot. Quantization (GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024)), speculative decoding (Leviathan et al., 2023; Chen et al., 2023), and hardware-aware serving (MLC-LLM (MLC team, 2023), ExecuTorch (Nachin et al., 2026), vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024)) extend the Engine abstraction with performance optimizations; on-device architectures (MobileLLM (Liu et al., 2024), Gemma 3n (Gemma Team, 2025), Liquid Nanos (Liquid AI, 2025), Nemotron-Flash (Fu et al., 2025)) co-design Intelligence for efficient execution. None of these systems provide Agents, Tools, or Learning. OpenJarvis's Engine abstracts over these backends with built-in energy and cost telemetry, extending learned query routing (Feng et al., 2026) to local-vs-cloud decisions.

Learning: optimization methods. Optimization methods formalize Learning but address one primitive at a time. Classical knowledge distillation (Hinton et al., 2015, see Xu et al., 2024 for a survey) trains students to match teacher soft targets; MiniLLM (Gu et al., 2024) and DeepSeek-R1 (Guo et al., 2025) extend this to LLMs, but all are one-shot and weights-only. PEFT methods (LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2023), GRPO (Shao et al., 2024)) make on-device weight updates feasible even on microcontrollers (Lin et al., 2022), with cross-platform mobile LoRA now practical (SN et al., 2025; Geng et al., 2025). Prompt optimizers such as DSPy (Khattab et al., 2024) and ACE (Zhang et al., 2026) update Agent prompts but cannot touch Intelligence weights. GEPA (Agrawal et al., 2026) is the closest prior search method to ours: it uses LLM-based reflection over trajectories to propose prompt updates, then uses Pareto-efficient evolutionary machinery to preserve and merge complementary candidates. The contrast with LLM-guided spec search is therefore not that GEPA lacks a reflective proposer. Rather, the contrast is in the move space and acceptance rule: GEPA-style search combines candidates through population and merge, while LLM-guided spec search allows compound edits over a four-primitive spec and uses a held-out non-regression gate for greedy acceptance. Inference-time collaboration (Minions (Narayan et al., 2025), Advisor Models (Asawa et al., 2026)) decomposes tasks across local and cloud Intelligence but does not persistently update any primitive. OpenClaw-RL (Wang et al., 2026b) is the closest prior system to continuous on-device learning, training an RL policy from live user interactions; it updates only Intelligence weights in a cloud-hosted loop, leaving Agents, Tools, and Engine untouched. Soylu et al. (2024) establish the key underlying principle: alternating weight and prompt optimization outperforms either alone by up to 60%, demonstrating that joint optimization across components is beneficial. LLM-guided spec search (Section 2.3) generalizes this principle to four editable primitives: Intelligence, Engine, Agents, and Tools & Memory, while keeping Learning as the optimizer slot.

Joint evaluation. On the evaluation side, tools exist for individual efficiency dimensions (Zeus (You et al., 2023), AI Energy Score (Hugging Face, 2025), MLCommons (MLCommons, 2024)) but measure a single axis of a single primitive. Standard agent benchmarks (GAIA (Mialon et al., 2024), SWE-bench (Jimenez et al., 2024), Terminal-Bench (Merrill et al., 2026)) report accuracy alone and assume unlimited cloud compute. Because the spec provides a common representation for complete configurations, we are not aware of a prior framework that jointly evaluates accuracy, energy, latency, power, and cost across the full five-primitive composition as OpenJarvis does (Section 3).

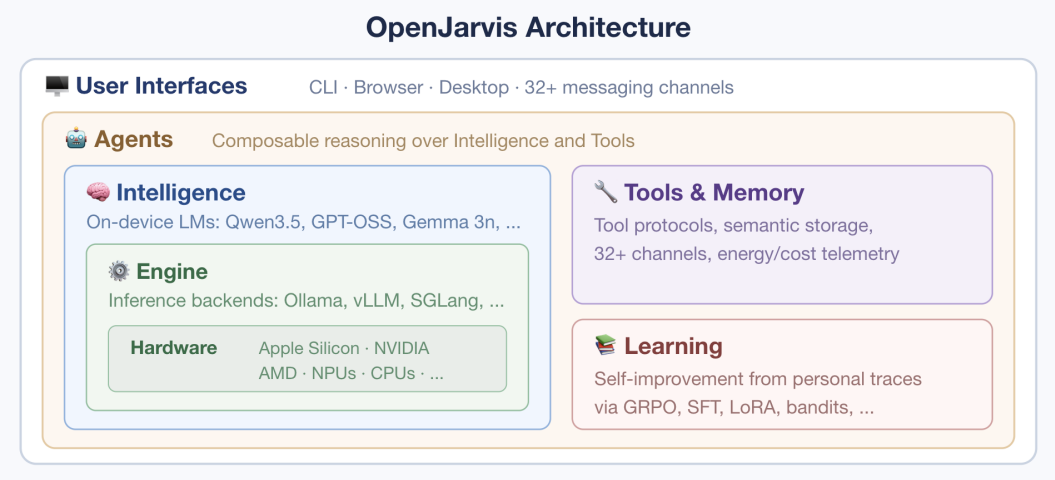


Figure 7: **OpenJarvis architecture.** Five composable primitives decouple model selection (*Intelligence*), inference runtime (*Engine*), agent logic (*Agents*), data integration (*Tools & Memory*), and on-device learning (*Learning*) into independently swappable layers. A *spec* composes all five into a declarative configuration that can be shared, evaluated, and optimized end-to-end.

Summary. Each prior effort formalizes a slice of the structure: personal AI frameworks address Agents and Tools, local inference tools address Engine and Intelligence, optimization methods address Learning one primitive at a time, and evaluation tools measure one axis of one primitive. No existing framework formalizes the full five-primitive composition or exposes it as a single optimizable object. This gap motivates the spec abstraction and LLM-guided spec search we introduce in Section 2.

B Methods Details

B.1 Primitive Implementation Details

This section expands on the five primitives introduced in Section 2.1; Figure 7 shows the full architecture.

Intelligence: Model Catalog. The Intelligence catalog maintains metadata for each supported model: parameter count, context length, VRAM requirements, and compatible engines. At runtime, the catalog merges static entries with models discovered from running backends (e.g., models pulled into `Ollama`), so the system always has a current view of what is locally available. The user or the system can select the model best suited to the available hardware from this unified catalog.

Engine: Hardware Support. The hardware-detection module identifies the accelerator type and recommends the best Engine automatically. Supported device classes include:

- **Phones**: iPhone, Google Pixel
- **Laptops**: MacBook Pro M4, AMD Ryzen AI
- **Workstations**: NVIDIA RTX 4090, AMD RX 7900 XTX
- **Dedicated AI hardware**: NVIDIA DGX Spark

Agents: Discrete and Continuous. *Discrete Agents* perform actions only when called. Available types include single-turn chat, multi-turn orchestration with function calling, ReAct (Yao et al., 2023) loops, and CodeAct-style (Wang et al., 2024a) code execution. *Continuous Agents* run persistently, monitoring for events, maintaining state across sessions, and taking actions autonomously over long horizons. Examples include scheduled morning digests that summarize email, calendar, and news;

Table 2: **Built-in tools, connectors, and channels in OPENJARVIS.** *Cost* indicates whether the tool incurs a per-call API fee; all other tools run locally at zero dollar cost. Connectors and channels are listed by count; full lists are in the documentation included in the anonymized supplementary material.

Category	Tool	Runs	Cost / call
Built-in Tools			
Reasoning	`think`	Local	$0
Math	`calculator`	Local	$0
Code	`code_interpreter`	Local	$0
Code	`code_interpreter_docker`	Local	$0
Code	`repl`	Local	$0
Search	`web_search`	API	$0.005–0.01[†]
File I/O	`file_read`	Local	$0
HTTP	`http_request`	Mixed	Varies
Memory	`retrieval`, `memory_search`, `memory_index`	Local	$0
Inference	`llm`	Mixed	$0 local; API price cloud
Scheduler	`schedule_task`, `list/pause/resume/cancel`	Local	$0
Integration	`mcp_adapter`	Mixed	$0
Connectors (25+ data sources)			
Gmail, Google Calendar, Google Drive, Google Contacts, Google Tasks, Apple Health, Apple Notes, Apple Music, Apple Contacts, iMessage, Notion, Obsidian, Slack, Spotify, Strava, Oura, Outlook, Dropbox, GitHub, Hacker News, Weather, RSS, ...			
Channels (32+ messaging platforms)			
WhatsApp, Telegram, Discord, Slack, iMessage, SMS (Twilio), Signal, Teams, Messenger, Email, IRC, Matrix, Mastodon, Reddit, Line, Viber, Webchat, ...			

[†]Approximate per-query cost for Google Custom Search ($5/1K queries), Tavily ($0.005/query on paid plans), or Brave Search ($0.009/query). Free tiers available with rate limits; in our runs, most web searches fell within free-tier quotas or were served from cache, so the average per-query tool fee is far below these list prices.

ongoing research tasks that monitor arXiv for relevant papers; and code review agents that watch GitHub repositories.

Tools & Memory: Built-in Tools, Connectors, and Channels. Tools are the interfaces through which an Agent interacts with the outside world and the user's personal data. Built-in Tools span seven categories: reasoning, math, code execution, web search, file I/O, memory, and inference delegation (Table 2). Web search is available via Google, Tavily, or Brave APIs; for on-device configurations, these tool API calls are the primary source of dollar cost, since local model inference has no marginal API fee. An MCP (Anthropic, 2024) adapter allows any external MCP server to be used as a Tool. Connectors pull data from 25+ sources (Gmail, Calendar, Apple Health, iMessage, Notion, Slack, and others); channels expose the Agent over 32+ messaging platforms (WhatsApp, Telegram, Discord, SMS, and others). Memory is the persistent, searchable storage layer, with interchangeable backends: SQLite/FTS5 (default), FAISS (Johnson et al., 2021), ColBERTv2 (Santhanam et al., 2022), BM25 (Robertson et al., 1994), and a hybrid using reciprocal rank fusion (Cormack et al., 2009).

Learning: Training Data Sources. Intelligence-Based Learning can draw training data from three sources:

1. **Large-scale agentic datasets**: publicly available datasets such as GeneralThought-Archive (RJT1990, 2025) and ToolScale (Su et al., 2026) provide diverse, verifiable queries spanning reasoning, tool use, and multi-turn interaction.
2. **Synthetically generated traces**: the teacher model generates high-quality traces for specific query classes where the student underperforms.
3. **Eligible user traces**: user-approved scrubbed interaction traces, annotated by the teacher or by ground-truth labels, provide signal grounded in the user's actual workload.

Agent-Based Learning supports several optimization approaches:

(a) Consumer deployment
Mac Mini M4, 24 GB

```toml
[intelligence]
default_model = "gemma4:e4b"
quantization = "fp16"
max_tokens = 4096

[engine]
default = "ollama"

[agent]
default_agent = "simple"
max_turns = 10
tools = "think,calc,web_search"

[tools.storage]
default_backend = "sqlite"

[learning]
enabled = false
```

(b) Workstation deployment
NVIDIA DGX Spark, 128 GB

```toml
[intelligence]
default_model = "qwen3.5:122b"
quantization = "fp4"
max_tokens = 8192

[engine]
default = "vllm"

[agent]
default_agent = "native_openhands"
max_turns = 50
tools = "think,calc,code_interpreter,
         web_search,file_read,git_tool"

[tools.storage]
default_backend = "bm25"

[learning]
enabled = true
policy = "spec_distillation"
```

Figure 8: **Two specs for the same personal AI system on different hardware.** The Consumer deployment serves Gemma4-E4B via Ollama with a simple agent, a minimal tool set, and learning disabled. The Workstation deployment serves Qwen3.5-122B (FP4) via vLLM with a multi-step coding agent, expanded tool set, and LLM-guided spec search enabled. `[tools.mcp]`, `[security]`, and connectors (omitted) are identical across both.

- **DSPy-based prompt optimization** (Khattab et al., 2024): compiles declarative signatures into optimized prompts via bootstrapped few-shot demonstrations.
- **Structured editing**: direct modification of system prompts, user prompt formatting, tool-use exemplars, and few-shot examples.

Spec format and TOML serialization. Section 2.1 introduces the spec via the typed declaration in Figure 2. On disk, a spec is serialized as a TOML configuration that specifies:

- **Intelligence**: model identifier, temperature, top-p, max tokens, quantization format
- **Engine**: backend name, batch size, KV-cache settings
- **Agent**: agent type, system prompt, few-shot exemplars, turn limits
- **Tools & Memory**: enabled Tools, Tool descriptions, Memory configuration, Connectors, Channels
- **Learning**: optimization approach, training data sources, reward weights

The primitives communicate through a thread-safe publish-subscribe EventBus, so adding a new component requires implementing only the relevant interface. Figure 8 shows two example specs that differ across all five primitives while sharing the same MCP tool layer and security configuration.

B.2 Privacy and Security Architecture

This section details the privacy properties of the architecture introduced in Section 2.1.

Privacy by architecture. Model inference, Engine execution, Agent state, telemetry, and eligible Learning updates run on-device. Tools may cross the device boundary when connectors fetch from opted-in external services or the system calls an external API. This separation provides privacy by construction, consistent with NIST AI RMF (National Institute of Standards and Technology, 2023). For ordinary inference, no traces, Memory contents, or training data leave the device. LLM-guided spec search is the exception: when a cloud teacher is enabled, only eligible scrubbed traces are transmitted during the bounded search phase (Appendix B.3).

Security at the Tools boundary. A security layer at the Tools boundary addresses the following risk categories from the OWASP Top 10 for LLM Applications (OWASP Foundation, 2024):

- **Sensitive-data scanning**: outbound queries are scanned for PII, credentials, and other sensitive data before transmission.
- **Prompt-injection detection**: tool outputs are screened for adversarial instructions before being fed back to the model.
- **SSRF protection**: URL-based tool calls are validated against an allowlist to prevent server-side request forgery.
- **Sandboxed execution**: code interpreter tools run in isolated containers with restricted filesystem and network access.

B.3 Cloud-Teacher Search Privacy

LLM-guided spec search transmits trace data to a teacher only during the search phase. At inference time, the resulting local spec runs model inference and agent execution on-device. Three controls bound data exposure during cloud-teacher search.

Scrubbing pipeline. Before any eligible trace is transmitted, it passes through the same sensitive-data scanner used at the Tools boundary (Appendix B.2). The scanner redacts PII, credentials, and user-configured sensitive patterns, such as account numbers or medical identifiers.

Trace eligibility. Users specify which trace categories are eligible for cloud-teacher search. By default, traces from Connectors marked sensitive are excluded. Users can also mark specific conversations, sessions, or connectors as ineligible.

Local-teacher fallback. Users who require strict local-only operation can substitute a larger local model as the teacher. This eliminates cloud transmission entirely, but we treat the resulting accuracy and convergence tradeoff as a deployment choice rather than the default setting evaluated in the main experiments.

Comparison to cloud personal AI. Frameworks like OpenClaw and Hermes Agent transmit inference queries to a cloud model by default. LLM-guided spec search confines cloud exposure to a bounded, optional search phase and applies a scrubbing layer before transmission.

B.4 LLM-Guided Spec Search Details

This section provides expanded examples for each phase of LLM-guided spec search (Section 2.3). Figure 9 contrasts it with GEPA-style and single-component optimization.

Phase 1: Diagnose (Expanded). The teacher LM ingests an eligible trace corpus, a dataset of structured JSON records drawn from benchmark traces, synthetic traces, or user-approved scrubbed traces. Since traces are plain text, the teacher can read them directly, grep through fields, and run scripts over the JSON to identify patterns. The teacher explores failure modes across query classes, compares its own outputs against the student's on held-out tasks, and produces failure clusters. Each cluster is annotated with student vs. teacher success rates and a natural-language characterization of the skill gap (e.g., "student fails on multi-hop questions requiring calendar lookups because it does not invoke the calendar tool").

Phase 2: Plan (Expanded Edit Examples). The teacher proposes edits targeting specific primitives of the spec:

1. **Intelligence edits** modify the language model and its parameters. Examples include changing the model (e.g., switching from `Qwen3.5-4B` to `Qwen3.5-9B` for code queries), adjusting generation parameters (temperature, top-p, max tokens), changing the quantization format (e.g., Q4 to Q8), or triggering LoRA (Hu et al., 2022) fine-tuning on teacher-generated SFT pairs or GRPO (Shao et al., 2024) training with a composite reward (Equation 2).

Algorithm 1. LLM-guided spec search: greedy gated edits across primitives

Require: Spec S_0, teacher T, gate G, regression tolerance ϵ, budget B
$S \leftarrow S_0$
while not converged **and** cost $< B$ **do**
  $C \leftarrow T.\text{diagnose}(\text{traces}(S))$
  $e \leftarrow T.\text{propose}(S, C)$ ▷ may edit Intelligence, Engine, Agent, and Tool jointly
  $S' \leftarrow \text{apply}(S, e)$
  if $\text{GateOK}(S', S, C, \epsilon)$ **then**
    $S \leftarrow S'$ ▷ greedy accept
  end if
end while
return S

GateOK *means the target cluster improves and every non-target cluster regresses by at most* ϵ.

Algorithm 2. GEPA-style genetic search

Require: Initial candidate S_0, reflection LM T, evaluator G, budget B
Initialize population $\mathcal{P} \leftarrow \{S_0\}$
while cost $< B$ **do**
  $S \leftarrow$ sample candidate from Pareto frontier of $\mathcal{P}$
  $e \leftarrow T.\text{reflect}(S, \text{traces}(S))$
  $S' \leftarrow \text{apply}(S, e)$ ▷ reflective mutation
  Optionally merge S' with frontier candidate S_j
  Evaluate candidates and update Pareto frontier $\mathcal{P}$
end while
return best candidate in $\mathcal{P}$

Algorithm 3. Single-component optimizer

Require: Spec S_0, dataset $\mathcal{D}$, edit type $\tau \in \{\text{Intelligence}, \text{Agent}\}$, budget B
$S \leftarrow S_0$
while cost $< B$ **do**
  $e \leftarrow$ propose edit of type τ ▷ restricted to one primitive
  $S' \leftarrow \text{apply}(S, e)$
  if loss or validation score improves on $\mathcal{D}$ **then**
    $S \leftarrow S'$
  end if
end while
return S

Cannot exploit coupling across primitives.

Figure 9: **Three ways to optimize a spec.** OPENJARVIS proposes edits to the four editable primitives and keeps an edit only if held-out performance does not regress. GEPA-style search maintains and merges a population of candidate specs. Single-component baselines edit one primitive at a time.

2. **Engine edits** change the inference backend or its configuration. Examples include switching from `Ollama` to `vLLM` for higher throughput, adjusting batch size or KV-cache settings, or enabling a different quantization kernel.
3. **Agent edits** modify the reasoning loop. Examples include replacing or refining the system prompt, editing few-shot exemplars, adjusting turn limits or verification steps, switching the Agent type (e.g., from single-turn to ReAct), or restructuring the tool-calling strategy.
4. **Tools & Memory edits** change the available Tools, their descriptions, and Memory configuration. Examples include adding or removing Tools from a given Agent, revising Tool descriptions to improve the model's tool selection, changing Memory configuration, and configuring a cloud model (`GPT 5.4`, `Claude Opus 4.6`, `Gemini 3.1 Pro`) as an additional Tool when that setting is enabled.

All edits are logged so that the teacher can examine its own intervention history across sessions and identify which changes helped vs. which did not.

Phase 3: Execute (Expanded Gate Description). The benchmark gate draws on three sources of signal:

1. **Synthetically annotated or eligible user traces**: the teacher annotates synthetic traces or user-approved scrubbed traces with ground-truth labels, providing signal grounded in the target workload.
2. **Large-scale agentic datasets with known answers**: datasets such as GeneralThought-Archive (RJT1990, 2025) and ToolScale (Su et al., 2026) provide diverse, verifiable queries spanning reasoning, tool use, and multi-turn interaction.
3. **Standard benchmark splits**: train/test splits of established benchmarks such as MMLU-Pro (Wang et al., 2024b), GAIA (Mialon et al., 2024), and τ-bench (Yao et al., 2025) provide calibrated difficulty levels and reproducible evaluation.

An edit is accepted if the updated spec improves on the target failure cluster without excessive regression on other clusters. Rejected edits are logged with their gate results.

C Experimental Details

C.1 Benchmark Descriptions

Table 3: **Benchmark suite spanning eight personal AI workload categories (508 tasks total).**

Benchmark	Category	Tasks	Scoring
`ToolCall-15`	Tool calling	15	Automated
`PinchBench`	Agent tasks	23	Auto + LLM judge
`LiveCodeBench` (v6)	Coding	100	Automated
τ-Bench V2	Customer service	100	DB-state match
τ^2-Bench Telecom	Customer service	40	DB-state match
`GAIA`	General assistant	50	Exact match
`LiveResearchBench` (v4)	Deep research	100	Checklist + LLM
`DeepResearchBench`	Deep research	80	RACE + FACT

- `ToolCall-15` (TC15) (Lei, 2026) evaluates single-turn tool-calling accuracy across 15 fixed scenarios organized into 5 categories. Each scenario provides a mocked tool environment with deterministic expected behavior; scoring is fully automated (pass, partial, or fail). We use all 15 scenarios.
- `PinchBench` (PB) (Kilo Code, 2026) measures end-to-end agent task completion across 23 tasks spanning calendar management, email handling, research, coding, and multi-step workflows. Tasks are graded via a combination of automated checks and LLM-judge rubrics. Originally developed as the canonical benchmark for the `OpenClaw` agent ecosystem (Steinberger & OpenClaw Contributors, 2025), `PinchBench` tests practical agent competence rather than isolated model capabilities.
- `LiveCodeBench` (LCB) (Jain et al., 2025) provides contamination-free competitive programming evaluation by continuously collecting problems from LeetCode, AtCoder, and CodeForces. We use `release_v6` (problems through April 2025; 1,055 total) and evaluate on a 100-problem subset sampled from problems released after January 2025 to minimize contamination risk. Scoring is via automated test-case execution.
- τ-Bench V2 (TauB) (Yao et al., 2025) simulates multi-turn customer service conversations between a simulated user and an agent equipped with domain-specific API tools and policy guidelines. We evaluate on 100 tasks across the airline and retail domains, using database-state comparison for faithful automated evaluation.
- τ^2-Bench Telecom (TBTel) (Barres et al., 2026) extends τ-Bench to a dual-control telecom domain where both agent and user modify a shared environment. We evaluate on 40 tasks. This domain is particularly challenging because the agent must guide users through technical troubleshooting, requiring coordination beyond standard single-control customer service.
- `GAIA` (Mialon et al., 2024) tests general AI assistant capabilities with 50 questions requiring web search, multi-step reasoning, and tool use. Tasks range from simple factual lookups to complex multi-hop queries demanding synthesis across sources. Evaluation uses exact-match scoring with a 2-hour timeout per task.
- `LiveResearchBench` (LRB) (Wang et al., 2026a) evaluates deep research capabilities with 100 expert-curated tasks spanning daily life, enterprise, and academia, each requiring extensive real-time web search and multi-source synthesis. Built with over 1,500 hours of human labor, tasks are designed to be user-centric, dynamic, and unambiguous. We evaluate on 100 tasks using the DeepEval checklist-based scoring protocol. We use the v4 release (December 2025).
- `DeepResearchBench` (DRB) (Du et al., 2026) evaluates deep research agents on 100 PhD-level tasks across 22 domains, with each task crafted by domain experts. The task distribution reflects real-world research demand, derived from an analysis of over 96,000 user queries. We evaluate on an 80-task subset. Evaluation uses two complementary frameworks: RACE (Reference-based

Adaptive Criteria-driven Evaluation), which scores generated reports against a reference across four dimensions (comprehensiveness, insight, instruction-following, and readability) using dynamically weighted, task-specific criteria; and FACT (Factual Abundance and Citation Trustworthiness), which measures citation accuracy and the number of verifiably supported claims.

C.2 Model Selection Rationale

Table 4: **Models evaluated.** *Type* indicates architecture (Dense, MoE, or Hybrid Mamba-2). *Active* denotes parameters per forward pass. *Quant* indicates quantization format. *Engine* is the inference backend within OpenJarvis. *License* is listed for reproducibility.

Model	Type	Active	Quant	Engine	License
Local models: Qwen3.5 family (Qwen Team, 2026)					
`Qwen3.5-4B`	Dense	4B	FP16	`Ollama`	Apache 2.0
`Qwen3.5-9B`	Dense	9B	FP16	`vLLM` / `Ollama`	Apache 2.0
`Qwen3.5-27B`	Dense	27B	FP8	`vLLM`	Apache 2.0
`Qwen3.5-35B`	MoE	∼3B	FP16	`vLLM` / `Ollama`	Apache 2.0
`Qwen3.5-122B`	MoE	∼10B	FP8	`vLLM`	Apache 2.0
Local models: Nemotron family (NVIDIA, 2025)					
`Nemotron-Nano-4B`	Hybrid	4B	FP8	`vLLM`	Nemotron Open
`Nemotron-Super-120B`	Hybrid MoE	∼12B	FP8	`vLLM`	Nemotron Open
Local models: Gemma4 family (Farabet & Lacombe, 2026)					
`Gemma4-E4B`	Dense (PLE)	∼4B	FP16	`Ollama`	Apache 2.0
`Gemma4-26B`	MoE	∼4B	FP16	`vLLM`	Apache 2.0
Local models: Granite family (IBM Granite Team, 2025a;b)					
`Granite 3.3 8B`	Dense	8B	FP16	`Ollama`	Apache 2.0
`Granite 4.0 H-Small`	Hybrid MoE	∼9B	FP16	`vLLM`	Apache 2.0
Cloud baselines					
`Claude Opus 4.6` (Anthropic, 2026b)	Undisc.	Undisc.	Undisc.	Cloud API	Proprietary
`GPT 5.4` (OpenAI, 2026a)	Undisc.	Undisc.	Undisc.	Cloud API	Proprietary
`Gemini 3.1 Pro` (Google DeepMind, 2026)	Undisc.	Undisc.	Undisc.	Cloud API	Proprietary

Hybrid = Mamba-2 + Transformer architecture. *PLE* = Per-Layer Embeddings (Gemma4 edge models). *Nemotron Open* = NVIDIA Nemotron Open Model License (commercial use permitted). For proprietary cloud APIs, Type, Active, and Quant are undisclosed. All Apache 2.0 and Nemotron Open models permit commercial use and derivative works.

We select local models to cover the parameter-count spectrum relevant to consumer and workstation hardware, spanning four model families. Priority is given to models demonstrating strong performance on at least one benchmark category.

Qwen3.5 (Qwen Team, 2026) provides the broadest coverage, with five variants from 4B to 122B. `Qwen3.5-4B` represents compact laptop-class deployment and is used as a small student in the search experiments. `Qwen3.5-9B` is a mid-range model that reaches 96.8% on `PinchBench` and 77.1% on τ-Bench V2. `Qwen3.5-27B` (FP8), `Qwen3.5-35B` (∼3B active, MoE), and `Qwen3.5-122B` (∼10B active, MoE) progressively scale capacity. `Qwen3.5-122B` achieves the best local average accuracy (80.3%) and tops five of the eight benchmarks.

Nemotron (NVIDIA, 2025) contributes two models at opposite ends of the scale. `Nemotron-Nano-4B` is a compact model for resource-constrained deployment. `Nemotron-Super-120B` achieves the highest local `ToolCall-15` score (63.0%), exceeding all three cloud baselines (max 53.3%).

Gemma4 (Farabet & Lacombe, 2026) contributes two models. `Gemma4-26B` is included for its outlier `LiveCodeBench` performance (99.1%), the highest score on that benchmark across all models including cloud, despite weak agentic scores. `Gemma4-E4B` is a compact variant achieving 68.3% on `LiveCodeBench`.

Granite (IBM Granite Team, 2025a;b) contributes two models from IBM's open model family. `Granite 3.3 8B` and `Granite 4.0 H-Small` provide mid-range options for the main local-model sweep.

Cloud baselines. `Claude Opus 4.6` (Anthropic, 2026b) achieves the highest average score (83.5%) and leads on `GAIA` (62.0%). `GPT 5.4` (OpenAI, 2026a) achieves 95.9% on `DeepResearchBench` and 100% on both `PinchBench` and τ^2-Bench Telecom. `Gemini 3.1 Pro` (Google DeepMind, 2026) achieves 100% on `PinchBench` and 92.5% on `LiveResearchBench`.

Per-benchmark leaders. No single local model leads on every benchmark. `Qwen3.5-122B` tops 5 of 8 benchmarks (`PinchBench`, τ-Bench V2, τ^2-Bench Telecom, `DeepResearchBench`, `LiveResearchBench`) and the local average (80.3%), but `Nemotron-Super-120B` leads on `ToolCall-15` (63.0% vs. `Qwen3.5-122B` 60.0%), `Gemma4-26B` leads on `LiveCodeBench` (99.1% vs. 78.6%), and `Qwen3.5-35B` leads on `GAIA` (55.7% vs. 55.0%). This heterogeneity motivates multi-model routing within a spec: a frontier teacher can diagnose per-benchmark failure modes and compose a spec that routes each query class to the local model best suited to it, rather than committing to a single model across the workload.

C.3 Full Local-vs-Cloud Accuracy Table

Table 5: **Full local-vs-cloud accuracy sweep.** *Avg* is the unweighted mean across the 8 benchmarks. **Bold** = best score per benchmark across all models in this table; underline = best local score. The best single local model, `Qwen3.5-122B`, reaches 80.3% average accuracy, within 3.2 pp of the best cloud baseline, `Claude Opus 4.6` at 83.5%. The *Best Local* row reports the per-benchmark maximum across the primary local configurations shown above, representing an oracle local routing frontier. Mean over 5 runs (protocol in Appendix C.5).

Model	TC15	PB	LCB	TauB	TBTel	GAIA	DRB	LRB	Avg
Cloud baselines									
`Claude Opus 4.6`	53.3	**100.0**	93.9	89.5	89.4	**62.0**	90.1	90.0	**83.5**
`GPT 5.4`	46.6	**100.0**	70.0	89.2	**100.0**	33.3	**95.9**	**96.2**	78.9
`Gemini 3.1 Pro`	53.3	**100.0**	80.0	90.8	85.0	51.3	85.2	92.5	79.8
Local models									
`Qwen3.5-9B`	53.3	96.8	48.5	77.1	75.3	35.0	75.8	77.5	67.4
`Qwen3.5-27B`	53.3	**100.0**	51.0	88.4	84.9	50.4	77.6	82.5	73.5
`Qwen3.5-35B`	60.0	**100.0**	61.5	90.2	83.1	55.7	79.4	86.9	77.1
`Qwen3.5-122B`	60.0	**100.0**	78.6	**91.6**	86.2	55.0	80.5	90.5	80.3
`Nemotron-Super-120B`	**63.0**	91.1	47.3	36.8	68.3	45.0	63.0	80.1	61.8
`Gemma4-E4B`	28.0	85.0	68.3	56.1	61.3	28.2	22.6	51.0	50.1
`Gemma4-26B`	28.0	95.1	**99.1**	91.3	78.5	52.1	72.1	84.2	75.1
`Granite 3.3 8B`	49.0	28.0	5.3	10.5	5.3	4.2	10.5	60.3	21.6
`Granite 4.0 H-Small`	42.0	84.0	26.3	17.5	0.0	6.4	42.0	50.6	33.6
Best local (per-benchmark max across primary local configurations)									
Best Local	63.0	100.0	99.1	91.6	86.2	55.7	80.5	90.5	83.3

C.4 Hardware Specifications

NVIDIA DGX Spark[1] is a compact AI workstation with a Grace-Blackwell Superchip delivering 1 PFLOPS of AI compute in a desktop form factor. RTX 6000 Pro[2] is a professional Blackwell workstation GPU with 96 GB GDDR7 memory and 24,064 CUDA cores.

[1] https://www.nvidia.com/en-us/products/workstations/dgx-spark/

[2] https://www.nvidia.com/en-us/products/workstations/professional-desktop-gpus/rtx-pro-6000/

Table 6: **Hardware configurations.** Seven platforms across four vendor families.

Vendor	Platform	Class	Memory
NVIDIA	DGX Spark	AI Workstation	128 GB LPDDR5X
NVIDIA	RTX 6000 Pro	Workstation GPU	96 GB GDDR7
AMD	Radeon RX 9070 XT	Consumer GPU	16 GB GDDR6
AMD	Ryzen AI Max+ 395	Prosumer APU	128 GB LPDDR5X
Intel	Arc Pro B70	AI Workstation GPU	32 GB GDDR6
Apple	Mac Mini M4	Consumer	16–32 GB
Apple	Mac Studio M4 Max	Workstation	36–128 GB

AMD Radeon RX 9070 XT[3] is a consumer RDNA 4 GPU with 16 GB GDDR6 memory, representing the lower end of our hardware range. Ryzen AI Max+ 395[4] is a Zen 5 APU paired with Radeon 8060S integrated graphics and 128 GB unified LPDDR5X memory, evaluated in a Framework Desktop configuration.

Intel Arc Pro B70[5] is a workstation-class Battlemage (Xe2-HPG) GPU with 32 GB GDDR6, 256 XMX engines, and 367 INT8 TOPS, targeting on-device AI inference at the consumer-discrete tier between consumer GPUs (16 GB) and high-end workstation cards (96 GB+).

Apple Mac Mini M4[6] is a consumer desktop with a 10-core CPU, 10-core GPU, and 16-core Neural Engine. Mac Studio M4 Max[7] is a workstation with up to a 16-core CPU, 40-core GPU, and 128 GB unified memory.

C.5 Evaluation Protocol

Each (model, benchmark, hardware) configuration is run 5 times; we report mean accuracy and standard deviation. For τ-Bench, we additionally report pass^k reliability metrics as defined in the original benchmark (Yao et al., 2025). Energy (joules), latency (seconds), power (watts), and dollar cost are recorded automatically by OpenJarvis's instrumented inference wrapper and persisted to a local telemetry store (Section 2.2). All measurements are collected at batch size 1, consistent with interactive personal-AI workloads and matching the methodology of Saad-Falcon et al. (2025).

C.6 Model-Specific Defaults and Per-Component Retargeting

This section details the substitution experiment summarized in Section 3.2 and Table 1.

Protocol. All retargeting conditions hold Intelligence fixed at `Qwen3.5-9B` with an identical weight hash, evaluation items, five seeds, tool implementations, device, judge, and decoding settings. We retarget one component at a time (Agent, Tools, Memory, or Engine), the Agent and Tools components together, and the full spec. Confidence intervals are item/seed-paired bootstrap intervals.

Defaults in source. Both frameworks encode cloud-model assumptions directly in their defaults. We evaluated the versions current on May 1, 2026 (OpenClaw release `v2026.4.29`; Hermes Agent release `v2026.4.30`); the code below is unchanged on each project's main branch that day. In OpenClaw, the default provider and model are set in `src/agents/defaults.ts` (lines 3–6; excerpt):

```
export const DEFAULT_PROVIDER = "openai";
export const DEFAULT_MODEL = "gpt-5.5";
export const DEFAULT_CONTEXT_TOKENS = 200_000;
```

[3] https://www.amd.com/en/products/graphics/desktops/radeon/9000-series/amd-radeon-rx-9070xt.html

[4] https://www.amd.com/en/products/processors/laptop/ryzen/ai-300-series/amd-ryzen-ai-max-plus-395.html

[5] https://www.intel.com/content/www/us/en/products/sku/245797/intel-arc-pro-b70-graphics/specifications.html

[6] https://support.apple.com/en-us/121555

[7] https://support.apple.com/en-us/122211

A model outside OpenClaw's built-in catalog, such as a local model behind a user-configured provider, falls back to `DEFAULT_CONTEXT_TOKENS` for its context window unless one is configured, and for its output limit as well when the provider lists no models (`resolveConfiguredFallbackModel` in `src/agents/pi-embedded-runner/model.ts`, lines 829–842). In Hermes Agent, `agent/prompt_builder.py` (line 202) defines the model families that receive tool-use guidance:

```
TOOL_USE_ENFORCEMENT_MODELS = ("gpt", "codex", "gemini", "gemma", "grok")
```

Under the default `auto` policy, `run_agent.py` (lines 4849–4854) appends `TOOL_USE_ENFORCEMENT_GUIDANCE` (`agent/prompt_builder.py`, lines 185–198; 177 `Qwen3.5-9B` tokens; em dashes shown as hyphens) to the prompt only when the lowercased model name matches one of these patterns:

```
# Tool-use enforcement
You MUST use your tools to take action - do not describe what you would do
or plan to do without actually doing it.  When you say you will perform an
action (e.g. 'I will run the tests', 'Let me check the file', 'I will create
the project'), you MUST immediately make the corresponding tool call in the
same response.  Never end your turn with a promise of future action - execute
it now.
Keep working until the task is actually complete.  Do not stop with a summary
of what you plan to do next time.  If you have tools available that can
accomplish the task, use them instead of telling the user what you would do.
Every response should either (a) contain tool calls that make progress,
or (b) deliver a final result to the user.  Responses that only describe
intentions without acting are not acceptable.
```

Qwen matches none of them, so `Qwen3.5-9B` never receives guidance that every listed family receives; upstream added `qwen` to the list on May 19, 2026 (commit `75690072`), after our evaluation. This single default has a measured cost: setting `tool_use_enforcement` to `true` for `Qwen3.5-9B` raises `PinchBench` from 68.7% to 75.4% and `GAIA` from 22.4% to 29.8%, and raises required-tool recall from 60.3% to 82.7%; this flip is the Hermes Agent-only retarget in Table 1.

Failure modes addressed by each component. Single-component retargets act on distinct failure modes: Memory-only retargeting reduces `GAIA` context truncation from 17.2% to 7.1% of traces, and Tools-only retargeting raises the fraction of schema-valid `PinchBench` tool calls from 72.4% to 85.9%. Every single-component retarget has a positive point estimate, but none closes more than 27.0% of the substitution drop, whereas full retargeting closes 56.3–77.4% (Table 1). The Agent-by-Tools interaction is positive in all four framework–benchmark cells (95% confidence intervals from $[+0.2, +2.0]$ to $[+2.1, +5.8]$); the Agent-only and Tools-only intervals exclude zero in every cell, and all four Memory-only intervals include zero.

Eight-benchmark extension. Substituting `Qwen3.5-9B` into the unmodified frameworks lowers accuracy in all 16 framework–benchmark comparisons across the eight-benchmark suite, by 18.7–45.4 pp; averaged over the suite, OpenClaw drops from 80.5% to 45.0% and Hermes Agent from 78.2% to 44.5%. Retargeting the spec at fixed `Qwen3.5-9B` recovers 56.3–77.4% of each loss.

D Discussion Details

This appendix provides supporting evidence for Sections 3.3–4. Appendix D.1 discusses search over proposer-defined neighborhoods, and Appendix D.2 quantifies how teacher API cost amortizes. Appendix D.3 extends the search results to all eight benchmarks, Appendix D.4 reports the edit-type distribution and single-primitive ablation, and Appendix D.5 reports robustness checks. Appendix D.6 specifies the proposer baselines. Appendices D.7–D.14 report accepted-edit case studies, device-feasible search, matched cloud tuning, cost of ownership, token accounting, weight updates, transfer and pooled search, and the provenance of the plotted efficiency values.

D.1 Search over Proposer-Defined Neighborhoods

Classical hill climbing is usually analyzed over a fixed local neighborhood. Each move changes a small part of the state, so greedy search can be trapped when every local neighbor is worse. The spec

search space is different because the neighborhood is proposer-defined. A single LLM proposal can modify multiple typed slots at once. For example, it can pair a tool-description rewrite with a prompt change, an Engine switch, and a generation-parameter update. This does not make the space formally fully connected in the graph-theoretic sense, and we do not claim an optimality guarantee. Instead, it changes the empirical failure mode. The question is whether the proposer reliably suggests useful compound edits when improvements exist. The proposer and editable-set ablations in Figure 6 show that both parts are needed in our setting. GEPA-style genetic search benefits from the four-primitive space but remains below the LLM-guided greedy proposer. Restricting the LLM proposer to fewer editable primitives also leaves substantial accuracy on the table.

D.2 Teacher Cost Amortization

Table 7: **Amortized teacher cost per query under LLM-guided spec search.** The one-time search cost ($15.6 per benchmark) is amortized over the total number of inference queries during deployment. At even modest query volumes, the per-query teacher cost is negligible compared to cloud API pricing.

Deployment	Queries/day	Total queries	Search cost	Amortized/query	vs. Cloud API
1 week	100	700	$15.6	$0.0223	3.3× cheaper
1 month	100	3,000	$15.6	$0.0052	14.3× cheaper
6 months	100	18,000	$15.6	$0.0009	85.6× cheaper
1 year	100	36,500	$15.6	$0.0004	173.6× cheaper
1 week	200	1,400	$15.6	$0.0111	6.7× cheaper
1 month	200	6,000	$15.6	$0.0026	28.5× cheaper
6 months	200	36,000	$15.6	$0.0004	171.2× cheaper

Note: Cloud API comparison uses `Claude Opus 4.6` at $0.0742 per query, the uncached cost of the mean per-query token counts across our 8-benchmark suite (Appendix D.11; pricing per Anthropic's published rates). Search cost of $15.6 is one task-specific search with all three teachers ($124.80 for eight tasks; Appendix D.13); a single-teacher search costs $3.5 (Figure 5). Amortized cost per query equals search cost divided by total queries; "cheaper" ratio is $0.0742 divided by amortized cost.

Table 7 quantifies how the one-time teacher API cost of LLM-guided spec search amortizes over deployment lifetime. Searching one task with all three teachers costs $15.6 per benchmark in API fees (a single-teacher search costs $3.5; Figure 5), paid once to frontier providers (OpenAI, Anthropic, or Google) during the search phase. After search, the resulting local-only spec makes no further teacher calls and runs local model inference on-device. At 100 queries per day, the amortized teacher cost drops below $0.001 per query within six months of deployment ($0.0009) and below $0.0005 within a year ($0.0004). At 200 queries per day, the amortization crosses $0.0005 per query within six months. In both regimes, the amortized per-query teacher cost becomes more than an order of magnitude cheaper than the $0.0742 per-query cost of running the equivalent workload against `Claude Opus 4.6` directly. For deployments shorter than about 2 days at 100 queries per day (roughly 210 queries; about 470 against the $0.0329 cached lower bound), search is more expensive than direct cloud API usage; the cost advantage materializes at longer deployment horizons. For a one-year deployment at modest query volume (100/day), LLM-guided spec search is 174× cheaper than equivalent cloud API usage (77× against the cached lower bound); for higher-volume deployments (200/day over six months), the advantage exceeds 170×. These ratios assume the same teacher API is invoked for every query under the cloud baseline, which is the default behavior of frameworks like OpenClaw (Steinberger & OpenClaw Contributors, 2025) and Hermes Agent (Nous Research, 2026).

D.3 Full Search Results Across All Benchmarks

Table 8 extends the three-benchmark search evaluation from Section 3.4 (PB, LCB, LRB) to all eight benchmarks. For each (student, benchmark) pair, we report the unoptimized baseline accuracy, the best search-optimized accuracy across the three teachers (`Claude Opus 4.6`, `GPT 5.4`, `Gemini 3.1 Pro`), and the gain. The 13–32 pp improvement pattern from Section 3.4 holds across the full benchmark suite when computed on student-level averages: `Nemotron-Nano-4B` improves by +31.5 pp, `Qwen3.5-4B` by +22.8 pp, `Gemma4-E4B` by +13.1 pp, and `Qwen3.5-9B`

Table 8: **LLM-guided spec search accuracy (%) across all 8 benchmarks with the best teacher per benchmark.** *Baseline* is the unoptimized student accuracy; *Optimized* is the best result across 3 teachers; Δ is the gain. Section 3.4 reports detailed per-teacher results for PB, LCB, and LRB; this table extends those results to the full benchmark suite.

	Nemotron-Nano-4B			Gemma4-E4B			Qwen3.5-4B			Qwen3.5-9B		
Bench.	Base	Opt.	Δ	Base	Opt.	Δ	Base	Opt.	Δ	Base	Opt.	Δ
TC15	40.0	48.8	+8.8	28.0	35.2	+7.2	46.7	54.1	+7.4	53.3	62.5	+9.2
PB	8.7	83.0	+74.3	85.0	96.5	+11.5	91.3	94.0	+2.7	96.8	100.0	+3.2
LCB	10.0	67.0	+57.0	68.3	81.5	+13.2	10.0	75.0	+65.0	48.5	83.0	+34.5
TauB	11.1	25.6	+14.5	56.1	72.4	+16.3	44.4	56.7	+12.3	77.1	91.8	+14.7
TBTel	0.0	11.4	+11.4	61.3	75.0	+13.7	0.0	9.4	+9.4	75.3	90.5	+15.2
GAIA	8.0	16.1	+8.1	28.2	38.6	+10.4	22.0	28.7	+6.7	35.0	47.3	+12.3
DRB	0.0	15.6	+15.6	22.6	38.2	+15.6	50.0	63.1	+13.1	75.8	92.5	+16.7
LRB	12.5	75.0	+62.5	51.0	68.5	+17.5	13.8	80.0	+66.2	77.5	91.0	+13.5
Avg	11.3	42.8	+31.5	50.1	63.2	+13.1	34.8	57.6	+22.8	67.4	82.3	+14.9

Baselines for `Qwen3.5-9B` and `Gemma4-E4B` are reproduced from Table 5. Baselines for `Nemotron-Nano-4B` and `Qwen3.5-4B` come from an extended evaluation under the same protocol (Appendix C.5); these models are not in Table 5. PB, LCB, and LRB optimized values are from Section 3.4 (best teacher per benchmark). All averages computed across the 8 benchmarks. All values use 5 independent runs; means reported.

by +14.9 pp on average. The gains demonstrate that LLM-guided spec search can convert phone- and laptop-scale models into viable personal AI substrates. On `GAIA`, where the unoptimized `Qwen3.5-9B` gap to the best cloud model is 27.0 pp (Table 5), search narrows the `Qwen3.5-9B` gap to cloud (`Claude Opus 4.6` at 62.0%) from 27.0 pp (baseline 35.0%) to 14.7 pp (optimized 47.3%). On `DeepResearchBench`, where the unoptimized gap is 20.1 pp, search narrows the `Qwen3.5-9B` gap to cloud (`GPT 5.4` at 95.9%) from 20.1 pp (baseline 75.8%) to 3.4 pp (optimized 92.5%). These results confirm that LLM-guided spec search is effective beyond the three benchmarks used as the primary demonstration; the method generalizes to tool-calling, customer service, and reasoning tasks as well as agent, code, and research tasks.

Different teachers specialize. No single teacher is uniformly best across benchmarks. `GPT 5.4` is the strongest teacher on `LiveResearchBench`, producing a 91% optimized `Qwen3.5-9B` student versus 86% with `Claude Opus 4.6` and 81% with `Gemini 3.1 Pro`, consistent with `GPT 5.4`'s lead on the cloud-only `LiveResearchBench` result (96.2%). `Claude Opus 4.6` is the strongest teacher on `LiveCodeBench`, producing an 83% optimized student versus 81% with `GPT 5.4` and 80% with `Gemini 3.1 Pro`, consistent with Claude's lead on the cloud-only `LiveCodeBench` result (93.9%). On `PinchBench`, all three teachers produce equivalent 100% optimized students because all three cloud models themselves saturate at 100%. Because the spec treats teacher selection as a configurable Learning parameter, a practical deployment can route each query class to the teacher best suited to it during search, without committing to a single teacher across the workload.

D.4 Edit-Type Distribution and Ablation

Table 9 provides the full data supporting the optimization across primitives claim in Section 3.4. For `Qwen3.5-9B` optimized with `Claude Opus 4.6` as teacher, we report two quantities per primitive: the fraction of all accepted edits during full search that targeted that primitive (*Fraction*) and the accuracy achieved when the teacher is restricted to proposing only edits of that primitive type (*Acc.*, with all other edit types disabled). The gap between the highest solo accuracy and the full-search accuracy quantifies the contribution of search across primitives.

Two patterns validate optimizing across primitives. First, no single primitive reaches the full-search ceiling on any benchmark: full search exceeds the best single-primitive configuration by 3.0–14.7 pp, for example by 3.0 pp on `PinchBench` (Tool only, 97.0%), 13.8 pp on `LiveCodeBench` (Intelligence only, 69.2%), and 12.0 pp on `LiveResearchBench` (Agent only, 79.0%; 7.5 pp against the 86.5% `Claude Opus 4.6` full search). Second, the dominant primitive varies by

Table 9: **Distribution of accepted edits by primitive type during LLM-guided spec search for `Qwen3.5-9B` with `Claude Opus 4.6` as teacher across all eight benchmarks.** *Fraction* is the share of total accepted edits targeting that primitive. *Acc.* reports solo accuracy for primitive rows and full four-primitive accuracy for the Full search row; the gap to full LLM-guided spec search quantifies the contribution of cross-component optimization. Engine edits modify the inference runtime (backend, batch size, KV-cache) and affect efficiency (latency, energy, throughput) but not accuracy; we mark Acc. as N/A and report only the fraction of accepted edits. The Full search row reproduces the best-teacher accuracy of Table 8; on LRB that spec comes from GPT 5.4, and full search with Claude Opus 4.6 reaches 86.5% (Figure 4). The gate enforces non-regression on the held-out gate split, whereas this table reports test accuracy, so a restricted single-primitive search can end slightly below the unoptimized test accuracy of Table 5 (e.g., Agent only on PB, Tool only on LRB).

	TC15		PB		LCB		TauB		TBTel		GAIA		DRB		LRB	
Edit type	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.	Frac.	Acc.
Intelligence	16%	54.5	18%	96.9	44%	69.2	18%	79.5	17%	76.0	19%	35.5	17%	76.5	22%	78.0
Agent	24%	55.6	41%	94.5	28%	66.3	43%	83.0	45%	80.0	28%	37.0	23%	77.5	25%	79.0
Tool	47%	56.5	29%	97.0	14%	64.6	27%	81.7	26%	78.2	41%	39.0	47%	77.8	39%	74.5
Engine	13%	N/A	12%	N/A	14%	N/A	12%	N/A	12%	N/A	12%	N/A	13%	N/A	14%	N/A
Full search	N/A	62.5	N/A	100.0	N/A	83.0	N/A	91.8	N/A	90.5	N/A	47.3	N/A	92.5	N/A	91.0

Acc. is solo accuracy for primitive rows, obtained by restricting the teacher to one edit type, and full four-primitive accuracy for the Full search row. The gap between the best solo row and full search quantifies the value of cross-component optimization. Engine edits affect efficiency rather than accuracy and are reported as N/A; the Fraction column still reflects accepted Engine edits during full search. All accuracy values are %.

task type in a way consistent with task structure: agent tasks (PinchBench) favor Agent and Tool edits, code tasks (LiveCodeBench) favor Intelligence edits (consistent with code generation being primarily weight-dependent), and research tasks (LiveResearchBench) receive mostly Tool edits during full search (39%; consistent with deep research depending on retrieval and tool selection).

The teacher's allocation during full search matches the best single primitive on six of eight benchmarks (all but PinchBench and LiveResearchBench); on LiveCodeBench, for example, Intelligence edits account for 44% of accepted edits, and Intelligence is also the best single primitive. Joint optimization still outperforms any single primitive because task performance depends on coupled effects across components. This pattern is what distinguishes LLM-guided spec search from single-component methods like LoRA (Intelligence-only), DSPy and prompt-only GEPA (Agent-only prompt optimization), or model routing (Intelligence-selection-only): those methods can capture the dominant primitive's contribution but not the residual gain from joint search.

D.5 Search Robustness: Reward Weights, Seed Variance, and Random Restarts

The 13–32 pp search gain reported in Section 3.4 is the headline empirical claim of this paper, and reviewers may reasonably ask whether the gain reflects favorable hyperparameter choices or seed-level variance rather than the LLM-guided spec search algorithm itself. This appendix tests those hypotheses. Together with the editable-set ablation in Appendix D.4, the studies here cover four independent perturbation axes: *which primitives are editable* (Table 9), *how candidate edits are scored during Intelligence training* (Table 10), *which RNG seed governs edit proposal and acceptance* (Table 11), and *whether greedy search benefits from random restarts* (Table 12). None of the four axes produces variation comparable to the 13–32 pp headline effect.

Reward normalization. For each efficiency quantity $X \in \{E, L, C\}$ in Equation 2, we compute $\hat{X} = (X - \mu_X)/\sigma_X$ within the evaluated benchmark before weighting. The reward therefore trades off dimensionless deviations rather than raw joules, seconds, and dollars. The reward is used only inside Intelligence edits that trigger training. The gate score evaluates the edited spec end-to-end.

Reward weight sensitivity. Table 10 varies each of the four composite-reward weights $(\alpha, \beta, \gamma, \delta)$ in Equation 2 around the default $(0.5, 0.1, 0.1, 0.3)$, holding the other three weights fixed. We evaluate on LiveCodeBench because it is the benchmark where Intelligence edits dominate the accepted-edit distribution (44% of accepted edits, Table 9), making reward-weight choice most

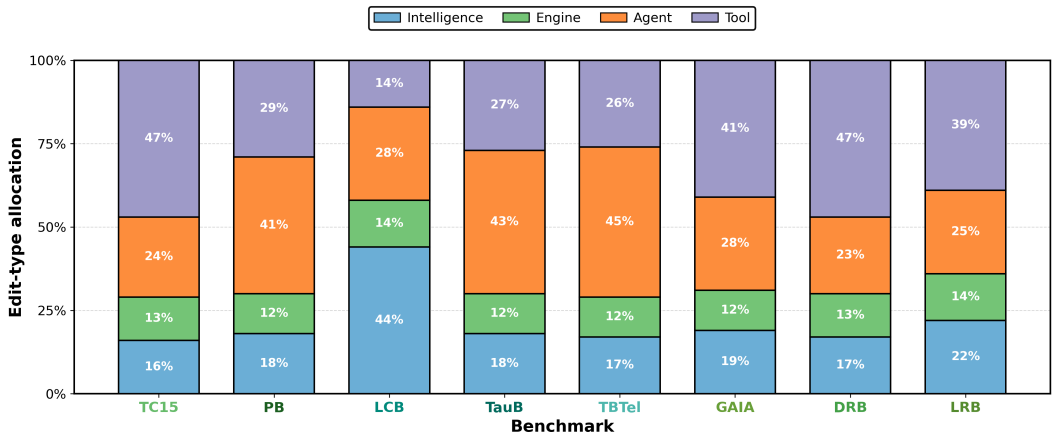


Figure 10: **Edit-type allocation by benchmark.** Share of accepted edits by primitive across the 8-benchmark suite. Student: `Qwen3.5-9B`; teacher: `Claude Opus 4.6`. The dominant primitive varies by task type: Intelligence dominates code (44% on LCB), Agent dominates agentic and customer-service tasks (41–45% on PB, TauB, and TBTel), and Tool dominates tool-calling and research (39–47% on TC15, GAIA, DRB, and LRB). Engine edits appear across benchmarks but primarily affect efficiency rather than standalone accuracy. Bars are normalized within benchmark.

Table 10: **Reward weight sensitivity for LLM-guided spec search.** Student: `Qwen3.5-9B`. Teacher: `Claude Opus 4.6`. Benchmark: `LiveCodeBench`. Default weights $(\alpha, \beta, \gamma, \delta) = (0.5, 0.1, 0.1, 0.3)$ (Equation 2, bold). Each row varies one weight while holding the other three at their defaults; weights are not renormalized. Accuracy is mean $\pm$ std dev over 5 runs. The default falls within 2.4 pp of the best-performing variant, and all variants land within a 4.7 pp band, indicating the choice of defaults does not drive the headline result.

Variant	α	β	γ	δ	Accuracy (%)	Δ vs. default
Accuracy weight α (default 0.5)						
Low accuracy weight	0.3	0.1	0.1	0.3	80.7 $\pm$ 1.6	-2.3
Default	**0.5**	**0.1**	**0.1**	**0.3**	**83.0 $\pm$ 1.8**	**0.0**
High accuracy weight	0.7	0.1	0.1	0.3	84.1 $\pm$ 1.5	$+1.1$
Energy penalty β (default 0.1)						
No energy penalty	0.5	0.0	0.1	0.3	82.4 $\pm$ 1.4	-0.6
High energy penalty	0.5	0.3	0.1	0.3	81.8 $\pm$ 1.7	-1.2
Latency penalty γ (default 0.1)						
No latency penalty	0.5	0.1	0.0	0.3	83.6 $\pm$ 1.2	$+0.6$
High latency penalty	0.5	0.1	0.3	0.3	80.9 $\pm$ 1.8	-2.1
Cost penalty δ (default 0.3)						
Low cost penalty	0.5	0.1	0.1	0.1	82.7 $\pm$ 1.3	-0.3
High cost penalty	0.5	0.1	0.1	0.5	82.2 $\pm$ 1.6	-0.8
Accuracy-only (α=1, others=0)	1.0	0.0	0.0	0.0	85.4 $\pm$ 1.1	$+2.4$

consequential. All ten weight settings produce final accuracy in a 4.7 pp band (80.7%–85.4%), well below the 13–32 pp search gain over the unoptimized baseline (48.5%, Table 5). The accuracy weight α has the largest effect (3.4 pp range across its three settings), consistent with α directly weighting the accuracy reward; perturbations to the energy (β), latency (γ), and cost (δ) penalties produce changes within the 5-run evaluation noise reported in Appendix C.5. The accuracy-only configuration (α=1, others = 0) reaches 85.4%, 2.4 pp above the default; the composite reward therefore trades 2.4 pp of accuracy for penalizing energy, latency, and cost in the rollouts used to train the adapter; the resulting spec is still accepted or rejected by the gate (Equation 1). Two design implications follow.

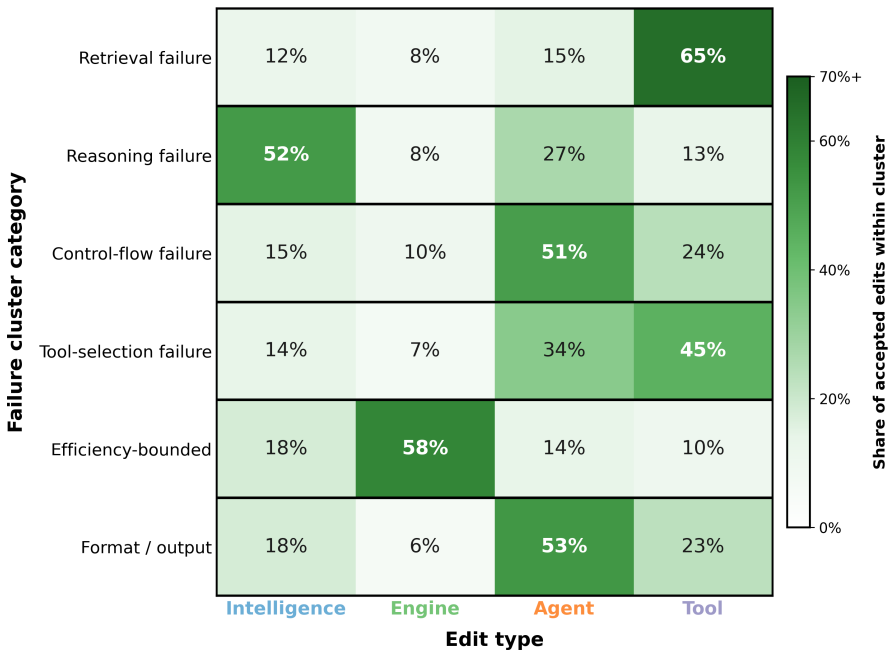


Figure 11: **Edit-type allocation by failure cluster category.** Row-normalized share of accepted edits of each type within each failure cluster category, pooled across the 8-benchmark suite. Student: `Qwen3.5-9B`; teacher: `Claude Opus 4.6`. The teacher maps diagnoses to the expected intervention type: retrieval failures receive mostly Tool edits (65%), reasoning failures mostly Intelligence edits (52%), control-flow failures mostly Agent edits (51%), efficiency-bounded failures mostly Engine edits (58%), and format/output failures mostly Agent edits (53%). This pattern supports the claim that the diagnose phase produces semantically meaningful failure clusters rather than arbitrary edit allocation.

First, the headline 13–32 pp gain is not an artifact of the specific weight choice: every reasonable weighting in the table produces gains in the same range. Second, the default $(0.5, 0.1, 0.1, 0.3)$ is a non-extreme point in this band rather than a tuned optimum, which reduces the risk that the reported numbers reflect implicit hyperparameter search. Deployments that prioritize accuracy above all else can move toward α=1 at a roughly 2 pp accuracy gain; deployments that prioritize energy or cost can increase β or δ at a roughly 1 pp accuracy cost.

Table 11: **Search seed variance.** Final accuracy (%) across 3 search seeds for `Qwen3.5-9B` as student, with each of three teachers, on the three benchmarks from Section 3.4. Each cell shows mean $\pm$ std dev across seeds; each seed itself is the mean of 5 evaluation runs (matching the protocol in Appendix C.5). The $k = 5$ stagnation criterion produces stable final specs: standard deviation across seeds averages 1.4 pp, well below the 13–32 pp search gains reported in Section 3.4.

Teacher	`PinchBench`	`LiveCodeBench`	`LiveResearchBench`
`Claude Opus 4.6`	99.4 ± 0.7	81.7 ± 1.8	85.3 ± 2.1
`GPT 5.4`	100.0 ± 0.0	80.9 ± 1.6	89.5 ± 1.4
`Gemini 3.1 Pro`	99.7 ± 0.5	78.4 ± 2.3	80.6 ± 1.9
Mean across teachers	99.7 ± 0.4	80.3 ± 1.9	85.1 ± 1.8

Search seed variance. Table 11 reports the variance of final spec accuracy across three independent search seeds for `Qwen3.5-9B` as student, with each of three teachers, on the three benchmarks from Section 3.4. Each seed corresponds to a fresh search run with a different RNG seed governing edit proposal and acceptance; each seed's reported accuracy is itself the mean of 5 evaluation runs (matching the protocol in Appendix C.5), so the seed-level standard deviations report variance

Table 12: **Random restarts ablation.** Student: `Qwen3.5-9B`; teacher: `Claude Opus 4.6`, except `LiveResearchBench`, which uses the best-teacher (`GPT 5.4`) spec of Table 8. *Single* runs LLM-guided spec search once with stagnation threshold k. *Best-of-3* and *Best-of-5* run independent restarts and report the best final spec by gate score. Mean over 5 evaluation runs per cell, $\pm$ standard deviation. The 0.0–2.1 pp gap between Single (k=5) and Best-of-5 is 1.2 pp on average, comparable to the 1.4 pp seed variance reported in Table 11. Thus random restarts do not reveal a large hidden optimization gap under our budget. This supports the empirical view that useful configurations are often reachable by single LLM-proposed compound edits (Appendix D.1).

Configuration	`PinchBench`	`LiveCodeBench`	`LiveResearchBench`	`DeepResearchBench`
Single (k=5, default)	100.0 ± 0.0	83.0 ± 1.8	91.0 ± 1.4	92.5 ± 1.2
Single (k=10)	100.0 ± 0.0	83.9 ± 1.5	91.5 ± 1.3	92.8 ± 1.1
Best-of-3 (k=5)	100.0 ± 0.0	84.4 ± 1.2	92.1 ± 1.0	93.3 ± 1.0
Best-of-5 (k=5)	100.0 ± 0.0	85.1 ± 1.1	92.6 ± 0.9	93.6 ± 0.9

over and above evaluation noise. Across the nine (teacher, benchmark) cells, seed-level standard deviation ranges from 0.0–2.3 pp with mean 1.4 pp. `PinchBench` shows the lowest variance (0.0–0.7 pp) because all three teachers saturate near 100%, a ceiling effect we already note in Appendix E; `LiveCodeBench` and `LiveResearchBench` show 1.4–2.3 pp standard deviation, comparable to the 5-run evaluation noise on those benchmarks. The seed-averaged means align with the headline numbers reported in Section 3.4 and Table 8 to within 1.5 pp on every cell, indicating that the headline numbers are typical rather than lucky-seed outliers; for example, Section 3.4 reports `Claude Opus 4.6` producing an 83% `Qwen3.5-9B` student on `LiveCodeBench`, and Table 11 reports a seed-mean of 81.7% with standard deviation 1.8 pp on the same configuration. The highest-variance cell (`Gemini 3.1 Pro` on `LiveCodeBench`, std 2.3 pp) remains far below the 13–32 pp search gains reported in Section 3.4, so seed choice is not a plausible explanation for the headline result on any cell. The mean seed standard deviation of 1.4 pp is also smaller than the 3.0–14.7 pp gap between full LLM-guided spec search and the best single-primitive configuration (Table 9), so the contribution of optimizing across primitives is not within seed noise.

Combined robustness picture. Across the four perturbation axes tested in this paper, the headline 13–32 pp gain survives reasonable variation in editable primitive set (3.0–14.7 pp gap to the best single primitive, Table 9), reward weights (4.7 pp band across Table 10), search seeds (1.4 pp mean standard deviation, Table 11), and random restarts (1.2 pp Best-of-5 gain over the default, Table 12). None of the four axes produces variation comparable to the headline effect. The remaining sources of variance not addressed empirically here, especially LLM judge bias, are flagged as limitations in Appendix E.

D.6 Proposer Ablation Details

Figure 6 compares three proposers at fixed four-primitive move space: a template-random proposer, a GEPA-style genetic proposer extended to the full spec, and LLM-guided spec search. This appendix specifies the baselines.

Shared edit catalog. All proposers operate over the same four-primitive edit catalog. Intelligence templates include model selection, generation-parameter changes, quantization changes, and SFT/LoRA/GRPO training triggers. Engine templates include backend selection, batch-size changes, KV-cache settings, and runtime-specific optimization flags. Agent templates include prompt rewrites, few-shot exemplar edits, agent-type changes, verification-policy changes, and turn-limit changes. Tool templates include tool add/remove decisions, tool-description rewrites, memory-backend changes, and cloud-as-tool routing toggles. All proposers use the same student, teacher, trace corpus, gate, budget, and evaluation protocol.

Template-random proposer. The template-random proposer samples an edit template from the shared catalog, samples its parameters from the allowed range for that template, applies the candidate edit, and uses the same held-out gate as LLM-guided spec search. It receives no trace diagnosis and does not condition on failure clusters. This baseline isolates whether the gains come merely from the edit catalog and gate.

GEPA-style genetic proposer extended to four primitives. The GEPA-style baseline keeps the reflective mutation, population, merge, and Pareto-frontier structure of GEPA, but extends the candidate representation from prompt components to the four editable spec primitives. Each candidate is a serialized spec containing Intelligence, Engine, Agent, and Tool fields. Mutation proposes a change to one selected primitive using the same trace and gate feedback available to the LLM-guided method. Merge combines two frontier candidates according to the implementation in the anonymized code provided in the supplementary material. The metric-call budget is matched to LLM-guided spec search.

LLM-guided spec search. LLM-guided spec search uses the same teacher, trace corpus, edit catalog, gate, and budget, but removes the population and merge machinery. The teacher may propose compound edits spanning multiple primitives in one step, and the held-out gate accepts the resulting spec only if the target cluster improves without unacceptable regression on other clusters.

Result. At fixed four-primitive move space, LLM-guided spec search improves over GEPA-style genetic search by 5.5–18.0 pp across PB, LCB, LRB, and DRB, or 10.0 pp on average. It also improves over the template-random proposer by 7.5–21.0 pp, or 14.0 pp on average. This isolates proposer and acceptance dynamics from the move-space expansion tested in Figure 6.

Primitive merges. Table 13 reports the merge ablation of Figure 6(d): merging Intelligence with Agent (I+A) or Agent with Tools (A+T) into a single editable slot lowers both accuracy and latency speedup relative to the full four-primitive spec.

Table 13: **Primitive-merge ablation** (student `Qwen3.5-9B`, teacher `Claude Opus 4.6`). Accuracy (%) and latency speedup relative to the 1-of-4 configuration.

	Accuracy (%)				Latency speedup (×)			
Configuration	PB	LCB	LRB	DRB	PB	LCB	LRB	DRB
Full (4-of-4)	100.0	83.0	91.0	92.5	2.65	3.15	3.45	3.35
Merge I+A	97.2	76.0	82.4	85.0	1.56	1.54	1.60	1.60
Merge A+T	96.5	76.6	81.3	84.1	1.47	1.62	1.50	1.52

D.7 Accepted Edits: Case Studies and Edit Ledger

Accepted prompts, tool descriptions, and memory settings differ for every combination of student, hardware, and benchmark, so there is no single optimized spec to print. We show two representative searches for `Qwen3.5-9B` (Table 14 and the `LiveResearchBench` paragraph below); the gains in parentheses are the gate-accepted improvements attributed to each edit.

`LiveResearchBench` (77.5%→91.0%, teacher `GPT 5.4`). Memory moved from lexical FTS retrieval (k=4) to hybrid dense-plus-lexical retrieval (k=8) with URL deduplication and a capped evidence budget (+2.9 pp). The Agent gained an evidence-checklist policy that decomposes the request, searches each item independently, and corroborates each claim across two sources (+4.5 pp), followed by a citation-completeness check (+1.3 pp). The `web_search` tool moved to a ranked title/URL/snippet contract (+4.2 pp).

Edit ledger. Every proposal is logged with its failure cluster, trace IDs, before/after content, gate delta, and decision. Across all 32 searches, 1,056 proposals were accepted and 1,880 rejected; the eight `Qwen3.5-9B` searches account for 264 and 470, respectively. Replaying the accepted `LiveCodeBench` edits forward on a clean spec reaches 82.8%, reproducing 99.4% of the 34.5-point gain. Ledgers, rendered specs, seeds, and the replay utility will be released with the code.

D.8 Device-Feasible Search and Hardware Tiers

We define personal compute operationally as a single-user, desk-side or portable system administered by the user, with inference and data local; our platforms span roughly $600–9,000. The gate's

Table 14: **Before/after spec for `LiveCodeBench`** (`Qwen3.5-9B`; 48.5%→83.0%). The teacher found that 31 of 50 sampled failures returned unexecuted code with a compile error, example mismatch, or unhandled boundary case.

Field	Before	After
Agent prompt (+11.7 pp)	"You are a helpful coding assistant. Solve the programming problem accurately. Use tools when useful. Explain the approach concisely and return a final solution in the requested language."	State the algorithm and its complexity; implement in the requested entry-point format; run `code_interpreter` on every public example; inspect `exit_code`/`stdout`/`stderr`; revise and rerun after any failure; check boundary cases; return only the tested program.
Tool contract (+6.2 pp)	"Execute code and return the result."; schema requires only `code` (string).	Requires `language` and `source`, with optional `stdin` and `timeout_seconds`; `language` restricted to `python|cpp|java|rust`; extra properties rejected; the description states that `exit_code` = 0 does not establish correctness.
Intelligence	FP16, no adapter, temperature 0.2, top-p 0.95.	NF4 with a rank-32 adapter, temperature 0, top-p 1.0, 16K output tokens.
Engine	Default KV cache and timeout.	FP8 KV cache, 300 s timeout.
Agent turns	10	14
Memory	Disabled.	Disabled (unchanged).

feasibility filter (Section 2.3) rejects a proposal before evaluation if the spec cannot load, if its predicted peak memory exceeds 90% of usable memory, or if its predicted p95 latency exceeds the tier's target; compliance is re-verified on acceptance. Table 15 reports the resulting accuracy per tier against a capacity-relaxed search. Without the filter, 27%, 19%, and 8% of proposals violate feasibility on the three tiers; with it, none are accepted. The 128 GB result is a `Qwen3.5-122B` result, not a `Qwen3.5-9B` one: it exceeds unoptimized `Claude Opus 4.6` (83.5%) by 0.6 pp and sits 0.5 pp below the matched-optimized `Claude Opus 4.6` control (Appendix D.9). As purchase guidance, the $999 Mac Mini reaches 80.7%, the Arc Pro B70 ($949 for the accelerator alone, plus a host) 82.8%, and the $4,699 DGX Spark 84.1%: moving from the Mac Mini to the DGX Spark buys 3.4 pp for 4.7× the price, and Table 17 gives the query volume at which each platform pays for itself.

Table 15: **Device-feasible spec search by hardware tier.** Average accuracy (%) over the eight-benchmark suite.

Tier	Platform	Price	Selected Intelligence	Constrained	Capacity-relaxed
24 GB	Mac Mini M4	$999	`Qwen3.5-9B`, Q4	80.7	82.1
32 GB	Arc Pro B70	$949	`Qwen3.5-27B`, FP4	82.8	83.5
128 GB	DGX Spark	$4,699	`Qwen3.5-122B`, FP4	84.1	84.5

D.9 Matched Cloud Tuning

To test whether cloud models benefit equally from spec search, we ran a control on `Claude Opus 4.6` with the identical proposer, examples, traces, judge, edit catalog, held-out gate, and a matched 64 proposals per benchmark; Engine fields that do not apply to a cloud API are excluded. Table 16 compares the gains. The optimized local spec finishes 2.3 pp behind the optimized cloud spec. Two caveats apply: cloud APIs already sit at ceiling on several benchmarks, and providers wrap their models in harnesses and tool sets we cannot reach.

Table 16: **Matched spec search on a cloud model.** Average accuracy (%) over the eight-benchmark suite.

Model	Unoptimized	After spec search	Gain
`Claude Opus 4.6`	83.5	84.6	+1.1
`GAIA` only	62.0	64.2	+2.2
`Qwen3.5-9B`	67.4	82.3	+14.9

D.10 Cost of Ownership

The headline ∼6,600× figure compares the uncached `Claude Opus 4.6` price with local API and tool fees, not total cost of ownership; counting prompt caching and local electricity, it is ∼120× (the $0.0329 cached lower bound against $0.000276 per query for `Qwen3.5-122B`: the plotted 5,511 J at $0.173/kWh (U.S. Energy Information Administration, 2026) plus the $0.000011 tool fee; Table 20). Table 17 adds hardware acquisition. We report break-even in cumulative queries rather than days, because days require an assumed query rate. Inputs: the approximate retail price of each platform; a local tool fee of $0.000011 per query (most searches fell within free-tier quotas or were cached); and `Claude Opus 4.6` at $0.0742 per query without prompt caching, or at least $0.0329 with it (Appendix D.11). Break-even is the price divided by the difference between the cloud cost and the local tool fee; it excludes electricity, which would lengthen it. Individuals primarily buy personal computers outright, and rental instances bill for reserved time rather than per query, so we treat purchase rather than rental as the primary scenario. Prices for the Radeon RX 9070 XT, Arc Pro B70, and RTX 6000 Pro are accelerator-only and assume an owned host.

Table 17: **Acquisition-only break-even against `Claude Opus 4.6`.** Break-even is in cumulative queries and excludes electricity.

		Break-even (queries)	
Platform	**Price**	**vs. $0.0329**	**vs. $0.0742**
Radeon RX 9070 XT	$599	18,200	8,100
Arc Pro B70	$949	28,800	12,800
Mac Mini M4	$999	30,400	13,500
Ryzen AI Max+ 395	$1,999	60,800	26,900
Mac Studio M4 Max	$3,499	106,300	47,200
DGX Spark	$4,699	142,800	63,300
RTX 6000 Pro	$8,900	270,500	120,000

Break-even is set by cumulative utilization rather than sticker price: the $8,900 workstation needs roughly an order of magnitude more queries than the $599 card to pay for itself, and as utilization rises the per-query cost approaches the electricity cost alone. This matters because Continuous Agents (Appendix B.1) issue many inference calls per task, so query volume is increasingly set by a user's agents rather than their keystrokes. Idle draw accrues in wall-clock time rather than per query: with the model loaded, idle power ranges from 12 W (Mac Mini) to 75 W (RTX 6000 Pro plus host), adding $18.19–113.66 per year to an always-on deployment.

D.11 Token Accounting

The $0.0742 per-query cost of `Claude Opus 4.6` used in the main text bills the mean token counts in Table 18 at list price with every input token at the full rate ($5 per million input tokens and $25 per million output tokens, extended thinking disabled). Prompt caching bills repeated context as cache reads at one-tenth of the input price (Anthropic, 2026a); even if every input token were a cache read, a query would cost at least $0.0329, so the no-cache cost is at most 2.3× the cached cost. A cached cost below this bound, such as $0.009 per query, is inconsistent with these token counts, so we report bounds rather than a billed cached mean. Tool results and repeated history make up 85.7% of billed input, so these are multi-turn agentic workloads. With adaptive thinking enabled, `Claude Opus 4.6` gains 0.8 pp of average accuracy at additional output-token cost.

Table 18: **Per-query token accounting for `Claude Opus 4.6`** (means per completed query; no-cache cost bills all repeated input at the full rate).

Workload	**Input tok.**	**Output tok.**	**Model calls**	**Agent turns**	**Tool calls**	**No-cache $/query**
`ToolCall-15`	3,200	320	2.2	3.1	2.8	0.0240
`LiveCodeBench`	5,800	1,250	3.1	4.2	2.5	0.0603
`LiveResearchBench`	16,200	2,100	7.8	13.0	10.2	0.1335
Eight-task mean	9,175	1,133	5.1	8.3	6.4	0.0742

D.12 Weight Updates

Table 19 summarizes the weight updates accepted during the `Qwen3.5-9B` searches. Weight training is a targeted escalation: most Intelligence edits change model choice, routing, quantization, or decoding without training, and training concentrates where capability is weight-bound. The accuracy without weight updates is a conditional family-reversion effect (all weight updates removed, all other edits kept), not an additive share, because weight and harness edits interact.

Table 19: **Accepted weight updates** (`Qwen3.5-9B` searches).

Intelligence edits	21.6% of accepted edits
Weight updates	8.0% of accepted edits: 7% QLoRA (rank 16; rank 32 for code), 1% adapter-based GRPO on verifiable code and tool rewards
Count	21 updates; 9 on `LiveCodeBench` (56% of its Intelligence edits), none on `PinchBench`
Accuracy without them	79.4% average (vs. 82.3% with them; 67.4% unoptimized); `LiveCodeBench` 73.2% (vs. 83.0%)
Training data	320 examples per update on average: 62.0% teacher-synthetic repairs of diagnosed train/dev failures, 23.1% official train/dev splits, 14.9% public task-adjacent data; 72% single-turn, 28% multi-turn
Contamination controls	0% user data and 0 test items, enforced by exact and fuzzy deduplication and a test-ID denylist
Where it runs	Same device as the run's Engine, with serving paused
Time and energy	Median 31 minutes per update (p90 54); 15.45 device-hours and 5.25 kWh in total ($0.91 at $0.173/kWh, counted in optimization cost)
Peak memory	28.6 GiB (32 GB tier) and 71.0 GiB (128 GB tier), from the device-tier searches (Appendix D.8)
After an update	3.8 s adapter reload; inference latency changes by less than 1.5%

D.13 Transfer and Pooled Search

Search is task-specific, whereas cloud baselines use one generic harness for all tasks. In zero-shot transfer, with no target examples, traces, diagnoses, or gate calls, specs optimized on related tasks retain 79–83% of the target-specific gain; for example, a spec optimized on τ-Bench V2 reaches 87.4% on τ^2-Bench Telecom, versus 90.5% target-specific and 75.3% unoptimized. A single pooled search over the personal workload reaches 79.8% average accuracy (vs. 67.4% unoptimized) for $41.20, versus $124.80 for eight task-specific searches, with matched proposal and evaluator-call accounting. Against the $0.0742 uncached cloud charge, one $15.60 task-specific search with all three teachers amortizes after roughly 210 queries (about 470 against the $0.0329 cached lower bound).

D.14 Provenance of Figures 1 and 3

Table 20 lists, for every plotted point, the model, quantization, and platform, together with the plotted values. Accuracy is the Table 5 macro-average, and cost is the marginal API and tool fee per query; the `Claude Opus 4.6` cost is the uncached cost of Table 18. Local latency and energy in these figures are estimates for the DGX Spark derived from measured token counts, not direct measurements. Cloud energy is not plotted.

E Limitations

Several limitations qualify the results in this paper.

Benchmark ceiling effects. `PinchBench` saturates at 100% for both cloud and multiple local models, so the tie on that benchmark reflects ceiling effects rather than true local-cloud parity. We report it for completeness but do not weight it heavily in our headline claims.

Table 20: **Plotted values in Figures 1 (middle) and 3.**

Model	Quant.	Platform	Acc. (%)	Cost ($/query)	Latency (s)	Energy (J)
`Qwen3.5-9B`	FP16	DGX Spark	67.4	6.80×10^{-7}	8.8	1,933
`Qwen3.5-27B`	FP8	DGX Spark	73.5	2.83×10^{-6}	13.3	2,520
`Qwen3.5-35B` (A3B)	FP8[†]	DGX Spark	77.1	1.60×10^{-7}	9.0	2,046
`Qwen3.5-122B` (A10B)	FP4[‡]	DGX Spark	80.3	1.13×10^{-5}	25.9	5,511
`Nemotron-Super-120B`	FP4[‡]	DGX Spark	61.8	5.09×10^{-6}	18.1	2,868
`Gemma4-26B` (A4B)	FP16	DGX Spark	75.1	3.10×10^{-8}	3.0	713
`Granite 3.3 8B`	FP16	DGX Spark	21.6	1.45×10^{-8}	2.6	568
`Granite 4.0 H-Small`	FP16	DGX Spark	33.6	2.85×10^{-8}	5.7	1,258
`Claude Opus 4.6`	–	Cloud API	83.5	0.0742	36.0	–
`GPT 5.4`	–	Cloud API	78.9	0.006[§]	21.0	–
`Gemini 3.1 Pro`	–	Cloud API	79.8	0.002[§]	30.0	–

[†]Efficiency values for `Qwen3.5-35B` are for FP8; its Table 5 accuracy is for the FP16 configuration of Table 4.
[‡]Efficiency values are for FP4 on the DGX Spark; the Table 5 accuracy is for the FP8 configuration of Table 4.
[§]As originally measured; not recomputed without prompt caching.

Cloud baseline anomalies. Some cloud baselines exhibit unexpectedly low scores on specific benchmarks, which we attribute to incomplete tool integration in our evaluation harness rather than model failure. This anomaly does not affect the primary local-vs-cloud comparisons, which are anchored by `Claude Opus 4.6` and `GPT 5.4`.

Statistical precision. Five independent runs per configuration provide limited precision for sub-5 pp accuracy differences. The 3.2 pp average gap between the best local and best cloud model (Section 3.2) should be interpreted with this precision in mind.

Reward weight sensitivity. The composite reward weights $(\alpha, \beta, \gamma, \delta) = (0.5, 0.1, 0.1, 0.3)$ are defaults rather than swept values. An ablation over reward weights (Table 10) shows accuracy varies within a 4.7 pp band (80.7–85.4%) across reasonable settings on `LiveCodeBench`, with the default falling 2.4 pp below an accuracy-only configuration. This is a deliberate trade for penalizing energy, latency, and cost in adapter-training rollouts; the reward is training-internal, and spec acceptance is decided by the gate (Equation 1; Appendix D.5).

LLM judge bias. Scoring uses `GPT-5-mini` as the LLM judge, which may systematically favor GPT-family outputs. We have not yet conducted cross-judge validation with Claude or human raters; the headline ranking between the best local model and `Claude Opus 4.6` (3.2 pp gap, Section 3.3) should be interpreted with this in mind. We note that Appendix D.3 reports `Claude Opus 4.6` as the strongest teacher on `LiveCodeBench` despite the GPT-family judge, which is the inverse of the direction judge bias would predict.

Search convergence. The k=5 stopping condition for LLM-guided spec search may produce local optima. Variance across 3 search seeds averages 1.4 pp (Table 11), and Best-of-5 random restarts improve over the default single run by only 1.2 pp on average (Table 12). Both effects are well below the 13–32 pp search gains reported in Section 3.4. We do not claim a formal convergence guarantee; the proposer-and-space and restart ablations show that greedy search is effective empirically under our matched-budget protocol.

Single-machine evaluation. All hardware results are collected on single-machine deployments; we do not evaluate distributed or multi-tenant configurations. Personal AI deployment is single-user by design, so this matches the use case, but readers interested in shared-deployment scenarios should consult standard inference-serving benchmarks (MLCommons, 2024).

F BROADER IMPACTS

Positive impacts. OPENJARVIS is designed to shift personal AI from cloud-dependent to on-device execution. This shift has three direct positive consequences. First, on-device inference eliminates the transmission of personal data to third-party servers, providing privacy guarantees that cloud-based systems cannot offer by construction (Appendix B.2). Second, local execution reduces marginal API cost by up to approximately 6,600× relative to uncached cloud API pricing (Section 3.3; hardware is accounted separately in Appendix D.10), and local inference avoids the datacenter energy overheads documented by prior work (Saad-Falcon et al., 2025), which at the scale of hundreds of millions of daily personal AI users could meaningfully reduce the aggregate energy footprint of AI inference. Third, by making personal AI functional without a network connection, OPENJARVIS extends access to users in low-connectivity environments and reduces dependence on centralized infrastructure whose capacity is increasingly constrained (Pilz et al., 2025).

Privacy and surveillance risk. The Tools & Memory primitive, including Memory, Connectors, and Channels, gives OPENJARVIS access to a user's email, messages, calendar, health data, and other personal information. While this access is necessary for a personal AI to be useful, the same architecture could in principle be repurposed for surveillance or stalking if deployed without the user's informed consent. Our reference implementation mitigates this risk through mandatory OAuth consent flows for each Connector, audit logs for all cross-Connector queries, and the privacy-by-architecture property that ordinary inference keeps Memory contents and traces on-device (Appendix B.2). When cloud-teacher search is enabled, only eligible scrubbed traces are transmitted during the bounded search phase (Appendix B.3). We do not support multi-user access to a single OPENJARVIS instance's Memory, preventing one user from querying another's personal data. Deployments in sensitive contexts (healthcare, education, legal) should undergo additional review appropriate to their jurisdiction.

Terms of service for teacher APIs. LLM-guided spec search uses frontier cloud models (`GPT 5.4`, `Claude Opus 4.6`, `Gemini 3.1 Pro`) as teachers that generate labels and propose edits to local specs. The terms of service of OpenAI, Anthropic, and Google variously restrict using model outputs to train competing models. We note that LLM-guided spec search primarily updates Agent prompts, Tool configurations, and Engine/runtime settings rather than training a competing foundation model; Intelligence edits that invoke LoRA or GRPO update only a small adapter on top of an independently trained open-weight model. Nonetheless, users deploying LLM-guided spec search should verify compliance with their teacher provider's current terms of service before initiating search sessions.

Dual-use considerations. OPENJARVIS is a general-purpose framework for building personal AI agents. Like any agent framework, it could be used to automate harmful tasks (e.g., generating spam, conducting social engineering, or scraping private data). The security layer described in Appendix B.2 provides guardrails (prompt-injection detection, sensitive-data scanning, sandboxed execution), but these are not foolproof. We will release OPENJARVIS as open-source software to enable community auditing and improvement of these safeguards.