BabyAGI Inside: Compiling a Bounded Agent Loop into Qwen Weights*

Yohei Nakajima
Untapped Capital

Abstract

We compile a bounded agent loop into the weights of an existing language model instead of running it as host code. BabyAGI Inside, a fixed-length, fixed-order simplification of BabyAGI, widens Qwen3.5-4B with a constructed controller, a task ledger, an in-model filesystem, and a halt condition; no parameter is trained, and after one objective prefill the host only samples tokens until the checkpoint emits HALT. The controller runs exactly as built: a teacher-forced oracle matches every compiled token, the merged model agrees with stock Qwen when the loop is inactive, and all 32 runs of a frozen live evaluation execute every phase, read back every file, and halt as specified. One inherited filesystem fault, which model text can trigger, remains unrepaired; it did not fire on Qwen3.5-4B. Compiling changes the cost, not any outcome we could detect: the compiled loop solves 28 of 32 tasks and the same protocol run as host code solves 27, with the same verdict on 31, but the compiled run takes about four times as long. Nor does the loop add measurable capability over plain chat: given the same token budget and made to answer, plain Qwen3.5-4B solves 29 and agrees on 31, and on the smaller Qwen3.5-2B it solves 12 to the loop's 8. On 48 longer held-out tasks (4 to 32 steps), a preregistered host-run screen of an adaptive version did worse: plain chat solves 38, the loop 25 even when stopped at the right step. What the construction provides is control: every failure we observed was the model's own content inside a loop that followed its schedule exactly.

1 Introduction

An LLM agent is usually implemented as two programs: a language model and host code that repeatedly prompts it, parses its outputs, invokes tools, updates state, and decides when to stop. The original BabyAGI script maintained a task queue, called a language model to execute and extend the queue, reprioritized tasks, and stored results outside the model (Nakajima, 2023). In that design the model generated text while the loop, queue, and memory remained host-side.

Recent work demonstrates that explicit programs can be compiled into transformer weights (Lindner et al., 2023; Shaw et al., 2025; Giannou et al., 2023). A separate line of systems work widens pretrained Qwen checkpoints with analytically constructed machines: Qwenputer adds a RISC-V processor; qwenFS adds a filesystem; and related projects add GPU, tensor-processing, and Linux-capable components (Agrawal, 2026b; Baraiya, 2026a;b; Bobliuuu, 2026; itsrealranky, 2026). In those systems Qwen remains in control: it decides when to use the compiled machine, typically by writing the tool-call syntax it was trained on, which the added weights then handle internally.

We study the reverse arrangement: can a bounded agent loop itself be compiled into model weights, with the pretrained language model serving as a subroutine called by the compiled controller? We make three claims, each tested separately:

*Generative language models were used as assistive tools during software development, experiment diagnosis, and manuscript editing. The author remains responsible for all claims and references.

1. **The controller runs exactly as built.** Scheduling, persistence, halting, and the ownership of every emitted token are fixed by construction, and we verify them offline with a teacher-forced oracle and live on a frozen 32-task suite. One inherited filesystem fault is the exception (Section 5).
2. **Compiling changes the cost, not any outcome we could detect.** The compiled loop and the same protocol run by host code reach the same verdict on 31 of 32 tasks, and the compiled loop takes about four times as long.
3. **The loop adds control, not measurable capability.** Given the same token budget and made to answer, plain chat solves as many tasks as the loop on Qwen3.5-4B and more on Qwen3.5-2B, and more than a host-run adaptive loop on longer tasks. What the loop adds is an exact boundary: every failure falls on the model's side, where the trace locates it.

The construction was first built on Qwen3.5-2B (Qwen Team, 2026b), which version 1 of this preprint reported (`https://doi.org/10.5281/zenodo.23966217`). Appendix A gives the three changes needed to port it to Qwen3.5-4B, and Appendix D summarizes the Qwen3.5-2B results.

2 Related work

Programs compiled into transformers. Tracr compiles RASP programs into the weights of new decoder-only transformers (Lindner et al., 2023), ALTA adds loops (Shaw et al., 2025), looped transformers can implement an instruction-set computer (Giannou et al., 2023), and Percepta compiles a WebAssembly interpreter into a small transformer (Percepta, 2026); each builds a model around the program. Tracr-Injection distills RASP algorithms into a pretrained model by optimization (Vergara-Browne & Soto, 2025). We add explicit weights to an existing checkpoint without training.

Constructed machines inside pretrained models. Qwenputer and qwenFS exploit Qwen3.5's recurrent Gated DeltaNet state (Yang et al., 2025) as machine state (Agrawal, 2026b; Baraiya, 2026a; Qwen Team, 2026a). Their descendants construct accelerators and operating-system components in widened residual lanes (Baraiya, 2026b; Bobliuuu, 2026; itsrealranky, 2026). The closest control-oriented construction is an in-weight oversight monitor that inserts and records a flag independently of the surrounding prompt, tamper-resistant against the model though not against whoever holds the weights (Agrawal, 2026a). BabyAGI Inside lifts constructed control from a single tool call or monitor to a repeated task workflow.

Internalized and program-controlled workflows. Agentic workflows can be internalized by fine-tuning, on generated procedural conversations (Dennis et al., 2026) or on agent trajectories (Chen et al., 2023). Neural Comprehension gates analytically compiled neural modules with a pretrained language model for symbolic operations (Weng et al., 2023). ReAct gives the language model control over interleaved reasoning and actions (Yao et al., 2023); LMQL constrains generation within an external program (Beurer-Kellner et al., 2023); and Agentic Programming argues that deterministic looping and sequencing should remain program-owned (Qi et al., 2026). We adopt that division of labor but place the program in the checkpoint. Budget forcing externally ends or extends a reasoning span (Muennighoff et al., 2025), and test-time compute can be allocated adaptively per problem (Snell et al., 2024). Our controller compiles a fixed reasoning budget into every execute phase, and we use host-side budget forcing for the plain-chat comparison.

3 System

BabyAGI Inside keeps the original division of labor: deterministic orchestration is program-owned, while planning and task execution are language-model-owned. The prototype simplifies BabyAGI to one up-front plan, fixed-order task selection, and a cycle count fixed when the checkpoint is built.

3.1 Construction

We start from the post-trained Qwen3.5-4B checkpoint (Qwen Team, 2026a) and the qwenFS merge at upstream baseline `fe3f5c0`, which prepends constructed filesystem layers and widens the residual stream

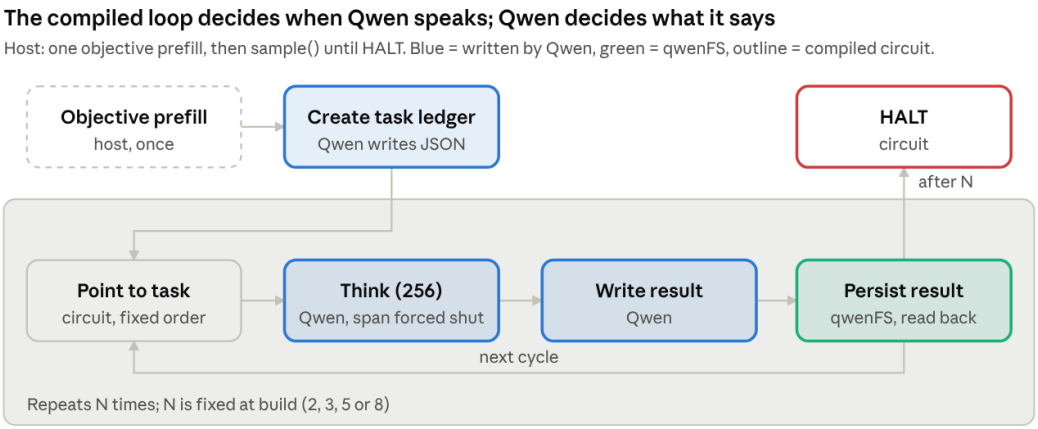


Figure 1: The ledger-pointer protocol. Blue boxes are model-owned generation spans, green is qwenFS, outlined boxes are compiled controller phases, and the dashed box is the host's one-time objective prefill. The controller fixes the task order but Qwen resolves each named ledger entry.

to 4,096 dimensions. The controller adds registers and a prompt ROM in its own dimensions. Each part reads and writes only its own lanes: Qwen's native dimensions, qwenFS's, and the controller's. Qwen's parameters are unchanged apart from the norm-gain rescaling that widening requires, and no gradient update is performed (layout in Appendix A).

The weights implement the controller and filesystem. The ledger and result files produced by a run live in inference state—the attention history and Gated DeltaNet state—and disappear when that run ends. Added control symbols encode START, HALT, and phase boundaries.

3.2 Ledger-pointer protocol

After the objective prefill and START, the controller executes the following protocol (Figure 1):

1. **Create.** The prompt ROM opens a `write_file` call and Qwen writes a JSON task ledger. qwenFS stores it.
2. **Prioritize.** qwenFS reads the complete ledger and the ROM names the next index ("execute ledger task k of N"). The controller never parses the task text; Qwen resolves the named entry from the ledger in context.
3. **Execute.** The ROM opens a 256-token reasoning span. The model fills all 256 positions, after which the controller closes the span and opens a result write. Qwen writes the result payload.
4. **Persist.** qwenFS stores `result.txt`; the following cycle reads it back.
5. **Halt.** After N cycles, the controller emits HALT.

For example, given "Start with the number 10. First multiply the result by 3; next add 7; then subtract 4; then double the result; finally add 7", Qwen writes a ledger with initial value 10 and the five operations, and the five-cycle checkpoint then executes them one per cycle, writing 30, 37, 33, 66, and 73. We build separate checkpoints for $N \in \{2, 3, 5, 8\}$; the loop does not choose its own stopping time.

3.3 Host interface

The host prefills the objective and Qwen tool schema once, with native thinking disabled, then performs greedy decoding until HALT or an external safety limit:

```
ids = prefill(objective) + [START]
while ids[-1] != HALT:
    ids.append(argmax(model(ids)))  # KV-cached in practice
```

The host does not schedule phases, parse model output, execute tools, repair malformed output, manage the ledger, or apply a logits processor after the objective prefill.

4 Evaluation

4.1 Checking that the loop runs as built

Three checks rule out different failures (Table 1). The **circuit oracle** teacher-forces a scripted run token by token and requires the checkpoint to predict every compiled token: prompt text, tool wrappers, filesystem responses, and control symbols. **Inactive parity** compares the merged checkpoint with stock Qwen3.5-4B on four text prefixes while the loop is inactive (top-1 agreement and KL divergence); it is a smoke check, not a proof of equivalence. **Live invariants** require, in every evaluation run, that all phases occur in compiled order (create, prioritize, and execute in each cycle; persist and halt are covered by the read and halt checks), that every file read equals the most recent write (the ledger in every cycle and the previous result in every cycle after the first, $2N - 1$ reads per run), that HALT follows the final cycle, and that every emitted token is attributed to exactly one owner.

4.2 Tasks and references

The frozen v3 suite has 32 tasks: two templates in each of four families (ordered literal transformations, sequential arithmetic, structured list operations, and checkable language edits) at chain lengths 2, 3, 5, and 8. A task is solved only if every intermediate result matches the reference exactly. The suite, prompts, decoding, and scorer were frozen before the port to Qwen3.5-4B was written, and the compiled sweep was preregistered with its checkpoint digests before its first task.

Two references answer two questions. **Does compiling change what the loop achieves?** The **host loop** runs the same protocol in Python around stock Qwen3.5-4B weights. It approximates the constructed control symbols and filesystem responses, so it is a reference rather than an exact simulator. **Does the loop help at all? Plain chat** gives stock Qwen3.5-4B the whole task in one thinking-enabled turn with the same prompt and scorer. Because the loop lets Qwen write at most N reasoning spans and $N + 1$ writes (the ledger and N results) of 256 tokens each, plain chat gets the same total: its thinking is closed after $(N + 1) \times 256$ tokens and it then has $N \times 256$ tokens to answer. This comparison was preregistered before any plain run. Appendix B also reports version 1's **one-turn** arm (a single ceiling of $N \times 256 + 256$ tokens).

All arms decode greedily in bf16 on one Apple M5. We report exact-chain counts with Wilson 95% intervals, and paired differences with item-bootstrap 95% intervals, whose preregistered reading is an improvement if the lower bound is above zero and a deficit if the upper bound is below it, together with the exact McNemar test (Appendix B).

5 Results: the controller runs as built

Every check passed (Table 1). Inactive parity is identical across cycle counts because the controller's cycle-specific weights are never reached, and no live run reached the 7,000-token safety limit (the longest emitted 5,675).

Ownership is exact: in every run Qwen chose 40–42% of the emitted tokens, almost all of them inside its 256-token reasoning spans, and the circuit forced the rest (Appendix B).

The exception. One component boundary is not yet guaranteed. qwenFS's parser, inherited unchanged from upstream, can enter write mode when the model's own reasoning imitates tool-call markup. On

Table 1: Verification of the four compiled Qwen3.5-4B checkpoints (2, 3, 5, and 8 cycles).

Check	What it rules out	Result
Circuit oracle	construction errors	6,280/6,280 compiled tokens
Inactive parity	damage to Qwen	top-1 4/4 prefixes; max KL 3.8×10^{-5}
Rebuild from the same inputs	nondeterministic builds	byte-identical, 4/4 checkpoints
Phases in compiled order	scheduling drift	32/32 runs (432 phases)
Reads equal the latest write	persistence errors	32/32 runs (256 reads)
HALT after the final cycle	runaway or early stops	32/32 runs
Every token has one owner	unaccounted output	32/32 runs

Table 2: Tasks solved on the frozen 32-task suite with Qwen3.5-4B, and wall time on one Apple M5.

Arm	Solved (95% CI)	Compiled − arm, points (95% CI)	Time
Compiled loop	28/32 (72–95%)	—	4.6 h
Host loop (same protocol)	27/32 (68–93%)	+3.1 (0.0 to 9.4)	1.1 h
Plain chat, same token budget	29/32 (76–97%)	−3.1 (−9.4 to 0.0)	1.2 h

Qwen3.5-2B this corrupted a read in 2 of 32 runs, both already failing tasks; on Qwen3.5-4B every read was exact, which shows only that the trigger was not reached: zero in 32 runs bounds the per-run rate below about 11% (Wilson 95% upper bound), and exact-read checks miss a dormant activation that is overwritten before a read. The repair, gating the parser on the real tool-call state, requires rebuilding every checkpoint (Appendix D).

6 Results: what the loop is worth

Compiling changes no outcome we could detect. The compiled loop and the host loop reach the same verdict on 31 of 32 tasks (Table 2); with one disagreement the data cannot tell them apart (exact McNemar $p = 1.0$). The one difference is a greedy branch after identical ledgers and 58 identical reasoning tokens, where only the host loop misread the task (Appendix B). Each arm ran once, greedily, and the host loop is itself approximate (on Qwen3.5-2B, fixing one omitted embedding moved it by two tasks; Appendix G), so one-task differences are within noise.

It is not cost-neutral. The compiled run took about four times as long as the host loop. It generates every one of its 100,545 tokens one at a time, more than twice the 46,800 plain chat generated, and 59% of them are text the circuit forces and a host would simply feed in. Its widened model decodes at 6.1 tokens per second against 11.1–11.3 for stock Qwen3.5-4B in the plain-chat runs. A compiled checkpoint's weights take 16.4 GB against 9.3 GB for stock Qwen3.5-4B. That is the price of a host that only samples.

The loop adds no measurable capability. Plain Qwen3.5-4B given the same token budget solved 29 tasks to the loop's 28 and reached the same verdict on 31; by the preregistered reading the difference is inconclusive (interval −9.4 to 0.0 points; exact McNemar $p = 1.0$), and the loop won no task that plain chat lost. The one difference is the task whose ledger Qwen filled with precomputed states instead of operations; answering directly, plain chat got it right. (Version 1's one-turn arm, 15 of 32, cut plain chat off mid-thought and measures that truncation; Appendix B.) On Qwen3.5-2B the loop did worse: plain chat at the same budget solved 12 tasks to its 8, winning four and losing none, a deficit by the interval rule (−25.0 to −3.1 points) though not by the exact test ($p = 0.125$). A fixed plan is only as good as the plan, and Qwen3.5-2B's were usable only for arithmetic (Appendix D).

Longer tasks widen the gap. Beyond eight steps, where one turn must carry every state in its own reasoning, the loop's per-step structure might pay off. A preregistered screen tested this before compiling

anything (Appendix C): 48 held-out tasks of 4 to 32 steps (listed chains, rules applied until a condition holds, and worklists), with the loop run as host code on stock Qwen3.5-4B, writing its own ledger and choosing after each step whether to stop. At the loop's token allowance plain chat solved 38; the loop solved 20, and 25 when stopped at the true length (-27.1 points, -39.6 to -14.6; 0 improved, 13 regressed), a deficit by the preregistered rule. Execution, not stopping, explains most of the gap: on worklists plain chat solved 16 of 16 and the loop 5, usually by dropping a queued job. It tests a host-run adaptive loop, not the compiled checkpoint.

Failures are the model's. All four compiled failures were content errors, at five or eight cycles, in runs that passed every check; two were the same edit ("uppercase only the word *lanterns*" produced `Lanterns`). The controller did exactly what the ledger said, so each trace locates the model's error (Appendix B).

7 Discussion

The compiled component owns phase order, persistence, and halting, and these behave exactly as built apart from the filesystem fault of Section 5; the model owns planning and content, where every failure we saw occurred. Compiling moves the program without changing what it achieves, at a cost in time and memory, and the loop adds nothing over a single turn with the same budget: on Qwen3.5-2B its fixed plans held the model back, and on longer tasks, where we expected its structure to help, a host-run adaptive version fell further behind. The case for the construction is control: an orchestration that cannot skip, reorder, or overrun its steps, and a trace that assigns every token and every failure to its owner.

That control has a specific scope. Constrained decoding could enforce the same phase structure, but its logits processor is host code; the compiled loop removes it, with the host's parser, scheduler, and halt decision, which helps only if the weights are trusted and the host is not. The price is patchability: a bug in the construction means rebuilding every checkpoint, as the parser fault shows. Fine-tuning a workflow into a model (Dennis et al., 2026; Chen et al., 2023) is far cheaper but guarantees nothing about which steps run.

Limitations. The loop length is fixed at build time and needs a separate checkpoint for each N; tasks run in fixed order; and Qwen plans once, unlike BabyAGI's incremental task creation. The host loop approximates the constructed embeddings, and inactive parity is a four-prefix smoke check. The suite has 32 short synthetic tasks, written before the port for a smaller model, and the 256-token reasoning budget was chosen on Qwen3.5-2B. The longer-task screen ran one adaptive design, as host code, on 48 generated tasks. All results use one model family, one backend, and one machine, with one greedy run per task.

Future work. Gating the parser on real call state and repeating every check; a recurrent program counter so the loop can decide when to stop; step formats that carry state as reliably as a single turn's reasoning; incremental task creation; and decoding forced spans in parallel, which could recover much of the cost.

8 Broader impact and research integrity

Compiling control flow into weights can make orchestration harder to inspect than ordinary source code and may create a false impression of determinism: the controller can be exact while model-owned content still activates an incompletely isolated parser. Releasing constructed checkpoints therefore requires explicit ownership traces, parser-state tests, model and compiler provenance, and a clear distinction between verified system invariants and empirical task accuracy.

9 Conclusion

A bounded agent loop can be compiled into an existing language model without training, and its controller then runs exactly as built, apart from one filesystem fault, while the host only samples. On Qwen3.5-4B it matched the same protocol run as host code, at four times the time, and did no better than plain chat at the same budget; a host-run adaptive version did worse on longer tasks. Its value is control, not capability.

References

Divyansh Agrawal. Building oversight tools into a model's weights. Technical blog post, 2026a. URL https://divagr.com/blog/building-oversight-tools-into-a-models-weights.

Divyansh Agrawal. Qwenputer. GitHub repository, 2026b. URL https://github.com/divagr18/qwenputer.

Vixhal Baraiya. qwenFS. GitHub repository, commit fe3f5c0, 2026a. URL https://github.com/vixhal-baraiya/qwenFS/commit/fe3f5c0dbebe5c827781e98ad839aecebcce32fa.

Vixhal Baraiya. qwenGPU. GitHub repository, 2026b. URL https://github.com/vixhal-baraiya/qwenGPU.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2023. URL https://arxiv.org/abs/2212.06094.

Bobliuuu. qwenTPU. GitHub repository, 2026. URL https://github.com/Bobliuuu/qwentpu.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning, 2023. URL https://arxiv.org/abs/2310.05915.

Simon Dennis, Rivaan Patil, Kevin Shabahang, and Hao Guo. Compiling agentic workflows into LLM weights: Near-frontier quality at two orders of magnitude less cost, 2026. URL https://arxiv.org/abs/2605.22502.

Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2301.13196.

itsrealranky. linuxformer. GitHub repository, 2026. URL https://github.com/itsrealranky/linuxformer.

David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2301.05062.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Yohei Nakajima. BabyAGI. GitHub repository, 2023. URL https://github.com/yoheinakajima/babyagi.

Percepta. transformer-vm: Compile programs directly into transformer weights. GitHub repository, 2026. URL https://github.com/Percepta-Core/transformer-vm.

Junjia Qi, Zichuan Fu, Jingtong Gao, Wenlin Zhang, Hanyu Yan, Xian Wu, and Xiangyu Zhao. LLM-as-Code: Agentic programming for agent harness, 2026. URL https://arxiv.org/abs/2606.15874.

Qwen Team. Qwen3.5-4B. Official model card and weights, revision 851bf6e806efd8d0a36b00ddf55e13ccb7b8cd0a, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-4B.

Qwen Team. Qwen3.5-2B. Official model card, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-2B.

Peter Shaw, James Cohan, Jacob Eisenstein, Kenton Lee, Jonathan Berant, and Kristina Toutanova. ALTA: Compiler-based analysis of transformers. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2410.18077.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Tomás Vergara-Browne and Álvaro Soto. Tracr-injection: Distilling algorithms into pre-trained language models, 2025. URL https://arxiv.org/abs/2505.10719.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. Mastering symbolic operations: Augmenting language models with compiled neural networks, 2023. URL https://arxiv.org/abs/2304.01665.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2412.06464.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

A Construction details and the port from Qwen3.5-2B

qwenFS owns residual dimensions 2048–3071, where it was designed to sit. Qwen3.5-4B's native residual is 2,560 wide, so it keeps dimensions 0–2047 and continues above qwenFS at 3072–3583, and the controller uses 3584–4023 to 3584–4059 depending on the cycle count; the remainder is padding. qwenFS prepends its layers to Qwen's 32 language-model layers. Extended embeddings encode file content and filesystem responses in qwenFS's lanes as well as Qwen's, and the control symbols carry a 0.25-scaled newline embedding in Qwen's lanes.

The port from Qwen3.5-2B needed three changes. *Residual placement*: Qwen3.5-2B's 2,048-wide residual occupies exactly 0–2047 (its controller uses 3072–3511 to 3072–3547), so the 4B native residual had to be split around qwenFS. *DeltaNet head map*: Qwen3.5-4B's Gated DeltaNet layers have 32 value heads over 16 key heads, each key head repeated across two value heads, so constructed filesystem head h writes value head $2h$ and the other value heads stay zero. *bf16 build*: Qwen's tensors are streamed one at a time into a bf16 model, because an fp32 allocation of the merged model would need an estimated 33 GB on the 32 GB build machine. Checkpoints built for Qwen3.5-2B with the ported code are byte-identical to the original Qwen3.5-2B builds, and the oracle's forced-token totals (840, 1,132, 1,716, and 2,592 for two to eight cycles) are the same on both models.

A Qwen3.5-4B build took 40–58 seconds with a peak memory footprint of 25.3–26.4 GB, and each checkpoint occupies 16.4 GB on disk. The suite's task wording and the protocol's 256-token reasoning budget were chosen on Qwen3.5-2B; no task was used to develop the port.

B Additional Qwen3.5-4B results

Statistics. Paired differences use 200,000 item-level bootstrap resamples (seed 20261105) and the exact two-sided McNemar test. Compiled minus host loop: 1 task improved, 0 regressed, $p = 1.0$. Compiled minus same-budget plain chat: 0 improved, 1 regressed, $p = 1.0$. The preregistered reading of each comparison uses the bootstrap interval. When only one or a few tasks disagree, resampling cannot produce a difference in the direction no task moved, so an interval bound of exactly zero reflects the method rather than evidence; the exact test is the better summary of such small differences. Two secondary budgets give the same verdict on every task: the one-turn ceiling with the answer guaranteed (F1; 38,771 tokens), and each task's compiled token total (F3; 65,425 tokens). Under both, plain chat solved 29, failing the same three eight-step tasks as the compiled loop and passing the fourth it failed (-3.1 points, -9.4 to 0.0; 0 improved, 1 regressed; $p = 1.0$). More budget than the one-turn ceiling changed nothing on Qwen3.5-4B.

One-turn arm. Version 1's plain arm gave stock Qwen one turn with a single ceiling of $N \times 256 + 256$ tokens. Every run reached the ceiling without closing its thinking, and 13 of 32 produced no parseable answer. It solved 15/32 (31–64%); compiled minus one-turn is +40.6 points (25.0 to 59.4; 13 improved, 0 regressed, $p = 0.00024$), a difference that measures the truncation.

By family. Table 3 gives exact chains and exact transitions (intermediate results scored along each arm's own trajectory).

Table 3: Qwen3.5-4B: exact chains and exact transitions by family (8 chains and 36 transitions per family).

Family	Compiled	Host loop	Plain, same budget	Plain, one turn
Ordered literal	6/8; 32/36	5/8; 24/36	7/8; 28/36	2/8; 10/36
Sequential arithmetic	8/8; 36/36	8/8; 36/36	8/8; 36/36	6/8; 32/36
Structured operation	8/8; 36/36	8/8; 36/36	8/8; 36/36	1/8; 8/36
Checkable language	6/8; 30/36	6/8; 30/36	6/8; 30/36	6/8; 20/36
Total	28/32; 134/144	27/32; 126/144	29/32; 130/144	15/32; 70/144

Plain chat solves one more task than the loop but four fewer transitions: its one ordered-literal failure missed all eight steps of an eight-step chain from the first, while the loop's two literal failures lost four steps between them. With eight tasks per family, these cells are descriptive only.

Token ownership. Table 4 reconciles the oracle's forced positions with one live arithmetic run per checkpoint; live counts differ because qwenFS echoes model-written content. Over all 32 runs, 100,545 tokens were emitted: 58,972 forced (39,616 ROM, 18,924 filesystem, 432 control symbols) and 41,573 chosen by Qwen (36,864 reasoning, 2,949 payload, 1,760 tool-call closure). The longest run emitted 5,675 tokens.

Table 4: Token ownership by checkpoint, from the arithmetic-add live runs.

Checkpoint	Oracle forced	Live forced (ROM/FS/ctl)	Qwen (reason/payload/close)	Total	Qwen share
2 cycles	840	693/181/6 = 880	512/35/30 = 577	1,457	39.6%
3 cycles	1,132	911/292/9 = 1,212	768/46/40 = 854	2,066	41.3%
5 cycles	1,716	1,347/548/15 = 1,910	1,280/66/60 = 1,406	3,316	42.4%
8 cycles	2,592	2,001/1,033/24 = 3,058	2,048/101/90 = 2,239	5,297	42.3%

Oracle and parity detail. All 4,608 model-owned reasoning positions in the oracle traces selected native-vocabulary tokens. Inactive parity per prefix: KL 4.5×10^{-6}, 3.8×10^{-5}, 1.0×10^{-7}, and 1.1×10^{-5}, identical on all four checkpoints.

The compiled–host disagreement. On the task "Begin with the exact text willow. First append -n3; next uppercase only its letters; finally prefix R3:", both runs wrote the same ledger and the same first 58 reasoning tokens of step one, ending `- Task 1: "append -n3"`. The compiled run continued the bullet ("this means append "-n3" to the current text") and wrote `willow-n3`. The host loop started a new paragraph, read the task as a flag ("append the number 3"), and wrote `willow3`, which every later step inherited. The contexts differ slightly because the host loop approximates the constructed embeddings, and greedy decoding diverged at that token; it is not a systematic advantage of compilation. The only other task whose results differ fails in both arms.

Failures and plans. The four compiled failures: two eight-cycle language tasks failed at step six on "uppercase only the word *lanterns*" (`Lanterns`); an eight-cycle literal task placed the final `!` inside a bracket at its last step; and a five-cycle literal task failed because Qwen's ledger listed precomputed states instead of operations, the third of which already carried a character from a later step. The host loop also failed all four, three of them identically. Qwen wrote a parseable JSON ledger on 32/32 tasks. The frozen plan metric, exact task strings and initial state, is 6/32, because Qwen3.5-4B usually paraphrases a step ("Sort the array in ascending numeric order." for "sort it in ascending numeric order") and then executes the paraphrase correctly: it solved all eight structured tasks with no exact structured ledger. A semantic operation-signature score, designed after observing Qwen3.5-2B ledgers and therefore post hoc, rates 21/32 ledgers executable.

Runtime. The compiled checkpoints decoded at 6.0–6.1 tokens per second at every chain length; the 32 live runs took 16,577 seconds. The host loop took 3,975 seconds: it generated 41,577 model tokens and fed in 39,616 forced prompt tokens without decoding them. Plain chat at the same budget took 4,228 seconds for 46,800 tokens; every forced answer parsed and none reached its answer cap. Of the 81,920 model tokens the same-budget allowance permits over the suite, plain chat used 46,800 and Qwen inside the loop chose 41,573. The one-turn arm took 3,985 seconds for 45,056 tokens.

C Longer tasks: a preregistered host-run screen

Design. The screen asks, before any such loop is compiled, whether a BabyAGI-style loop that decides for itself when to stop beats plain chat on tasks longer and less specified than the v3 suite. Its preregistration fixed the arms, caps, scoring, three predictions, and a decision rule before any task was decoded, and the preregistered analysis computed the decision. A seeded generator wrote a 48-task test split and a 16-task

development split, both frozen by hash before any model saw them. The test split has 16 tasks of each kind. *Long chains* list every operation, over v3's four families at 8, 16, 24, and 32 steps. *Rule-until* tasks apply one rule until a stated condition holds (4 to 28 steps, not stated). *Worklists* give a job table whose jobs add follow-up jobs to a queue under a first-in, last-in, or alphabetical rule, with state `{"done":[...],"queue":[...]}` (4 to 20 steps, not stated). The 48 tasks have 738 steps in all. Scoring is the v3 scorer: every intermediate state must match, and the number of results must equal the true length L.

Arms. The loop runs as host code around stock Qwen3.5-4B, greedily in bf16: the host advances the stages, holds the files, and feeds the scaled control-symbol embedding before each stage, as the host loop does. Qwen first writes a ledger holding an initial state and either a numbered `tasks` list, when the objective lists its operations, or a `rule` and an `until` condition (at most 600 tokens). Each cycle then reads the latest result and the ledger, opens a 256-token reasoning span and a result write of at most 240 tokens, and asks a one-token decision restricted to continue or stop; stopping, or the end of cycle 40, ends the run. The prompt names each cycle's number but never the length. The *adaptive* arm uses Qwen's decision; the *stopped-at-L* arm forces continue before L and stop at L, a fixed-length loop. *Plain chat* thinks for at most $600 + 256L$ tokens, is closed as in the main comparison, and answers in at most $241L$; together these equal the loop's maximum model-token allowance at L. Its prompt states no length, and its last JSON results list is scored whatever its length. A development check on the 16 development tasks could revise prompt wording up to three times. One revision, to the stop question, was adopted by a dated amendment before any test run; nothing else changed.

Results. Table 5 gives each arm. The preregistered decision is NO-GO: the stopped-at-L loop's interval against plain chat lies below zero, so the adaptive loop is not worth compiling for capability. Paired over the 48 tasks, adaptive minus plain chat is -37.5 points (-52.1 to -25.0; 0 improved, 18 regressed; $p = 7.6\times10^{-6}$), stopped-at-L minus plain chat -27.1 (-39.6 to -14.6; 0 improved, 13 regressed; $p = 0.00024$), and adaptive minus stopped-at-L -10.4 (-18.8 to -2.1; 0 improved, 5 regressed; $p = 0.0625$). The preregistered analysis (10,000 resamples, seed 0) gives the same readings; its first interval ends at -22.9. Neither loop arm solved a task plain chat failed.

Table 5: Longer-task screen, Qwen3.5-4B: tasks solved by kind (16 each), exact steps (of 738), and cost.

Arm	Solved (95% CI)	Long chain	Rule-until	Worklist	Exact steps	Model tokens	Hours
Loop, adaptive stop	20/48 (29–56%)	9	6	5	391	204,117	6.3
Loop, stopped at L	25/48 (38–66%)	9	11	5	453	215,813	6.5
Plain chat, same budget	38/48 (66–88%)	10	12	16	598	236,289	6.1

Where the loop loses. *Stopping* costs five tasks, all rule-until. The adaptive loop stopped exactly at L on 20 of the 26 tasks whose steps were all correct when it decided (the preregistered threshold for compiling was 90%); over those correct states its decisions were right 344 times in 350. *Execution* costs most of the rest. In 9 of the 11 worklist failures in each loop arm, the first wrong state has the right `done` list but drops a job that should have stayed queued: processing `fern` from the queue `[iris, fern]`, the loop wrote `[birch, reed]`, losing `iris`. The development runs showed the same error. Every execute step used all 256 of its reasoning tokens, so whether a larger per-step budget would help is untested, and it would change the matched budget. Plain chat closed its own thinking on 5 of 48 tasks, and it failed six exact-string long chains and four rule-until tasks of 10 to 28 steps.

The adaptive loop on the original suite. On the 32 v3 tasks the adaptive loop solved 17, against 28 for the compiled loop (-34.4 points, -50.0 to -18.8; 0 improved, 11 regressed), 27 for the host loop, and 29 for plain chat at the same budget. The loss is in the ledger. Every v3 task lists its operations, so the prompt asks for a numbered task list, and with one the loop solved 15 of 17; it wrote a rule instead for eight tasks and failed all eight, and seven ledgers did not parse (two of those solved). The compiled loop's fixed length and ledger pointer remove both choices, so the screen measures this adaptive design, not the compiled loop's protocol at longer lengths.

Predictions. The three preregistered predictions that could fail held. Plain chat was force-closed on most tasks (43 of 48). Stopping at L solved at least as many tasks as the adaptive stop (25 against 20, losing none). Plain chat's per-step exact rate fell faster with step index than the stopped-at-L loop's (87%, 79%, 66%, and 74% in 8-step bands, against 59%, 57%, 74%, and 71%), though only longer tasks reach the later bands, so the bands mix task with position.

D The Qwen3.5-2B build

Version 1 of this preprint (`https://doi.org/10.5281/zenodo.23966217`) reported the Qwen3.5-2B build; the repository contains its records and the documents of each step from Qwen3.5-2B to Qwen3.5-4B. In brief: all 32 runs executed every phase and halted with exact accounting, the oracle matched every forced token on all four checkpoints, and inactive parity was top-1 4/4 with a maximum KL of 4.1×10^{-4}. The compiled loop solved 8 of 32 tasks, all sequential arithmetic; the host loop solved 7, and the one-turn arm 5, with 24 of its 32 runs producing no parseable answer. Plain chat at the same budget solved 12: compiled minus plain is -12.5 points (interval -25.0 to -3.1; 0 improved, 4 regressed; exact McNemar $p = 0.125$), a deficit by the preregistered interval rule but not significant by the exact test. With the one-turn ceiling but a guaranteed answer (F1), plain chat solved 9 (-3.1 points, -9.4 to 0.0; 0 improved, 1 regressed). Given each task's compiled token total (F3), plain chat solved 15 (-21.9 points, -37.5 to -9.4; 0 improved, 7 regressed; $p = 0.016$), a deficit by both the interval rule and the exact test. Guaranteeing an answer at the one-turn ceiling thus raised plain chat from 5 to 9, and more budget raised it to 12 and then 15: the original three-task gap was truncation, and with budget plain Qwen3.5-2B pulls ahead. Under no budget, on either model, did the compiled loop solve a task plain chat failed.

Wrong plans executed faithfully. In the two eight-cycle structured tasks, Qwen3.5-2B's ledger combined the objective's first operations into one addition and then repeated an add-one task. Every result executed the corresponding model-written task (8/8 step fidelity) while none matched the reference (0/8): planning failed while execution and orchestration succeeded.

Parser fault. In two eight-cycle literal runs a read returned the latest write plus extra text (78 bytes became 1,086 in one run and 45 became 80 in the other). The inherited qwenFS rule recognizes a file-text start pattern without checking that a tool call is open, so tool-like markup in Qwen's reasoning switched the filesystem into write mode until the next genuine tool-call end. Both runs continued through every later phase and halted, and both were already failing, so no task verdict changed. The repair gates text-start on the real tool-call state.

Reasoning budget and earlier protocols. The protocol's 256-token budget was chosen with a native-weight diagnostic: stock Qwen3.5-2B given each of 144 gold task–state pairs reached 50.0%, 42.4%, 54.9%, 79.9%, and 87.5% single-step accuracy at 16, 32, 64, 128, and 256 reasoning tokens (Figure 2). An earlier typed protocol let Qwen prioritize with exactly one legal token through an added residual coordinate, moving both legal logits at least 1,893 above the best illegal logit, but Qwen chose the first task in both orders of a two-probe diagnostic, and with no reasoning span it solved 0/12 held-out tasks. Few-shot exemplars and a model-maintained ledger lifecycle also failed; these motivated the reasoning span and the fixed pointer.

E Reproducibility and artifact map

The repository (`https://github.com/yoheinakajima/babyagi-inside`; project page `https://babyagi.yohei.me`) contains the frozen benchmark, the live records for both models, the host-loop and plain-chat records, trace exports, the source needed to build and score the protocol and to regenerate the budget-forced plain arms, and CPU-only statistics scripts that reproduce every number in this paper. The suite hash is `e880e043f7ac412aaab938036cccd5f79305aed4cd0c13a7204e09fe07c1dff4`, and the Qwen3.5-4B sweep's experiment hash is `4a96e7aab95cc0db341a35281fe4a5562eef92ead69ecd16c39fd9ebf6491a29`. Qwen3.5-4B is model-hub revision `851bf6e806efd8d0a36b00ddf55e13ccb7b8cd0a`; its file hashes are in the repository, and the upstream qwenFS baseline is `fe3f5c0`. Checkpoint weights are not included. They are rebuilt locally

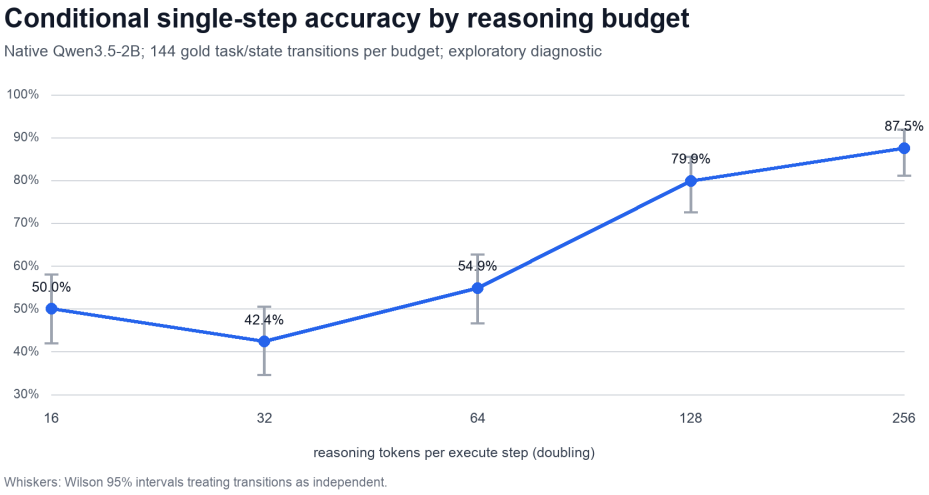


Figure 2: Qwen3.5-2B: single-step accuracy versus bounded reasoning budget on 144 gold task/state pairs per budget. Whiskers are Wilson 95% intervals treating transitions as independent.

from the pinned files, and each rebuild must reproduce the tree SHA-256 recorded in the repository (for Qwen3.5-4B: `bbc5aa8b...`, `edbceb5f...`, `8ab06101...`, and `913f236d...` for two, three, five, and eight cycles). All decoding is greedy. The supplement contains each preregistration document and its configuration, for the Qwen3.5-4B sweep, the corrected parity check, the plain-chat arms (F1, F2, and F3, all specified before any plain run), and the longer-task screen, with a lock record giving each lock's commit, time, and document SHA-256, its amendments, and the time of the experiment's first run. Every document hashes to its recorded lock, and each experiment's earliest lock precedes its first run; the screen's one amendment, the stop-question wording from its development check, precedes all of its test runs. The screen's records, its frozen task splits (test split SHA-256 `ab3e35d7...`), its analysis output, and the harness that produced them are in the repository, and a CPU-only script re-scores every record against its task. The lock commits are in a private repository, so their times are self-reported.

F Changes since version 1

Version 1 of this preprint reported the Qwen3.5-2B build only. This version reports the port to Qwen3.5-4B as the primary evaluation, adds the same-budget plain-chat comparison on both models, and adds a preregistered screen on longer tasks (Appendix C). It also corrects three errors in version 1:

- **Inactive parity.** Version 1 reported a maximum KL divergence of 0.0019 for Qwen3.5-2B. The parity check built its reference model on the meta device and materialized it with uninitialized non-persistent buffers, so the reference's rotary frequencies were zero and its full-attention layers saw no positions. Against a correctly initialized reference, the per-prefix KL divergences are 5.7×10^{-5}, 1.2×10^{-4}, 3.7×10^{-6}, and 4.1×10^{-4} on each of the four checkpoints. Checkpoints, live decodes, and baselines load the model normally and are unaffected; checkpoints built with the corrected code are byte-identical to the evaluated ones.
- **Controller dimensions.** Version 1 gave 3072–3208, the span of an early three-cycle baseline build. The evaluated Qwen3.5-2B checkpoints use 3072–3511 (two cycles) to 3072–3547 (eight cycles).
- **Plain baseline.** Version 1 described the one-turn plain arm's budget as strong and compared the compiled loop's 8/32 with its 5/32. Every one-turn run exhausted its budget before closing its thinking. Against plain chat at the same budget, the compiled Qwen3.5-2B loop is behind (Appendix D).

G Superseded analyses

The initial host-loop analysis on Qwen3.5-2B reported 9/32 tasks. It omitted the compiled control symbol's nonzero view in Qwen's native dimensions. Adding the 0.25 newline-embedding surrogate reduced it to 7/32 and removed two structured false positives; the original result is withdrawn but retained in the audit artifacts. A 32-transition pilot budget curve also contained a sampling bug and was excluded before the complete 144-transition comparison.