Autonomous Hyperparameter Search Under a Fixed Five-Minute Budget on a Consumer GPU

Eric Rhea, Astal Claw

Independent researchers

March 22, 2026

Abstract

We present a 35-run autonomous hyperparameter search campaign using the Windows/RTX fork of `autoresearch` on a single NVIDIA RTX 3080 (10 GB), trained on TinyStories under a fixed five-minute wall-clock budget per experiment, followed by a 17-run post-search confirmation sweep across 2–4 seeds for the most important configurations. Starting from an initial baseline of 1.027091 validation bits per byte (val_bpb), the search reached a best single-run result of 0.495600, a 51.7% improvement over the original baseline and a 35.2% improvement over the pre-overnight starting point. In the follow-up confirmation sweep, the final recommended configuration averaged 0.497631 val_bpb versus a confirmed baseline mean of 0.996635, a 50.1% improvement. The strongest robust gains came from reducing total batch size and shrinking model depth; the standalone attention-pattern change was weak in follow-up and is therefore treated as inconclusive rather than decisive. The broader practical lesson is that a consumer GPU like the 3080 is most useful not as a miniature frontier-training machine, but as a fast local search instrument for discovering compute-fit recipes.

Keywords: autonomous experimentation, hyperparameter optimization, consumer GPUs, TinyStories, compute-fit

1. Introduction

Autonomous experimentation is increasingly moving from proof-of-concept demos toward usable research instruments. Recent agentic systems have shown that LLM-driven workflows can propose ideas, write code, run experiments, visualize outputs, and even draft papers (Lu et al., 2024). Older automated machine-learning work already established the basic premise: if search is adaptive and compute-aware, it can outperform static human guesswork under tight budgets (Jaderberg et al., 2017; Li et al., 2016; Li et al., 2018).

The more useful question is whether automation can produce credible results inside a constrained, inspectable setting. That is why this report deliberately focuses on a single RTX 3080, a five-minute training budget, one training file, one benchmark, and an auditable keep/discard loop. The aim is not open-ended scientific automation; it is to test whether a bounded autonomous loop can still extract material performance gains under commodity constraints.

TinyStories makes that question experimentally tractable. Eldan and Li (2023) showed that tightly scoped synthetic story data can support coherent language-model behavior even in very small regimes, making the benchmark suitable for low-cost experiments. That matters because it means short runs are not meaningless noise; they can still provide real signal about training dynamics and model fit. In parallel, compute-aware scaling work argued that optimal behavior depends on how compute is allocated, not just on parameter count or raw architectural scale (Kaplan et al., 2020; Hoffmann et al., 2022).

This paper therefore asks a narrower operational question: what configuration works best for the Windows/RTX `autoresearch` path on a single RTX 3080 when every candidate is given only five minutes of training? Our answer is not a new scaling law or a new optimization algorithm. It is a dense empirical note showing that a weak but legible autonomous loop can still discover a materially better local optimum and leave behind a clean experimental ledger.

2. Related Work

2.1. 1 Small-model language modeling and constrained benchmarks

TinyStories is the most important benchmark citation for this report because it legitimizes the basic experimental surface. Eldan and Li (2023) showed that when the distribution is constrained enough, very small language models can still produce coherent multi-paragraph English. That makes TinyStories

more than a toy; it is a cheap but meaningful substrate for studying small-model training behavior under bounded budgets.

DistilBERT broadens that story from benchmark design to compact-model practice. Sanh et al. (2019) demonstrated that smaller, faster transformer models can retain much of the usefulness of larger systems via distillation, reminding us that efficiency gains do not come only from hyperparameter tuning. In the context of our paper, DistilBERT acts as a useful contrast: our overnight run improved a small-model training recipe without using teacher-student compression at all.

Taken together, these papers justify the paper's basic legitimacy claim: a narrow low-resource regime can still support informative model-behavior research. They also keep us honest about scope. TinyStories makes the setup scientifically tractable, but it also makes it local; DistilBERT reminds us that there are parallel efficiency strategies we did not explore.

2.2. 2 Attention and architecture choices

The Transformer paper remains the architectural foundation for any discussion of attention-pattern choices. Vaswani et al. (2017) established full self-attention as a practical, high-performance alternative to recurrent and convolutional sequence models, and that baseline matters here because our earliest architectural signal was that **full attention appeared to beat the mixed sliding-window pattern** on this hardware budget.

That empirical signal becomes more interesting, not less, when placed next to Longformer and BigBird. Longformer introduced a local-window plus global-token attention pattern designed to scale better on long documents (Beltagy et al., 2020), while BigBird showed that sparse attention patterns can preserve useful theoretical properties while reducing the cost of full attention (Zaheer et al., 2020). In other words, the mixed/local attention family is not a gimmick; it is a serious design space.

This matters for interpretation. Our result does **not** show that sparse or local attention is bad in general. It shows that, on this 3080, under this five-minute budget, the full-attention setting was worth testing and initially looked promising, but its standalone advantage weakened in confirmation. This is exactly the kind of local result that needs literature around it; otherwise it is too easy to misread as either trivial or overclaimed.

2.3. 3 Scaling laws, compute fit, and optimization dynamics

Scaling-law work established the broader intuition that model quality follows regular relationships with model size, data, and compute (Kaplan et al., 2020). Chinchilla sharpened this further by showing that under a fixed budget, "bigger" is not automatically better; optimality depends on how compute, capacity, and training exposure are balanced (Hoffmann et al., 2022). Our setup is much smaller, but the same philosophical move applies: the right model is the one that fits the budget horizon, not the one that merely looks large and impressive on paper.

The batch-size story in this paper also fits into prior optimization work. Smith et al. (2017) argued that batch size and learning-rate schedules jointly shape optimization behavior, implying that update cadence itself is a first-class training variable. Our overnight search found exactly that kind of local effect: shrinking total batch size from 2^19 down to 2^15 produced the single cleanest monotonic gain curve in the campaign.

Sharpness-Aware Minimization adds another useful lens. Foret et al. (2020) argued that better generalization can come from optimizer dynamics that prefer flatter solutions rather than merely lower training loss. We did not test SAM directly in this report, but it is relevant because it widens the explanation space: our optimizer wins may not be only about scalar hyperparameter pressure, but more generally about how the optimization path uses a small amount of compute to land in better regions of parameter space.

2.4. 4 Budget-aware hyperparameter optimization

The strongest weakness in earlier drafts was that they talked about autonomous search without enough serious HPO literature around it. That gap is now easier to name. Bayesian optimization provides a canonical black-box search foundation for expensive machine-learning objectives (Snoek et al., 2012). Hyperband framed HPO as a resource-allocation problem rather than a static search problem, using aggressive early stopping to explore many candidates cheaply (Li et al., 2016). ASHA pushed that logic into an asynchronous controller that makes better use of wall-clock time under parallel workloads (Li et

al., 2018). BOHB combined multi-fidelity scheduling with model-based acquisition to produce a stronger guided-search baseline (Falkner et al., 2018).

More specifically, Freeze-Thaw Bayesian Optimization and Fabolas are highly relevant to the kind of loop we ran here. Freeze-Thaw BO models partially observed learning curves so that runs can be paused and resumed intelligently (Swersky et al., 2014). Fabolas exploits dataset-size fidelity to gather useful information much faster than fully expensive evaluations would allow (Klein et al., 2016). Multi-fidelity GP bandit optimization provides the more formal theoretical story for cheap approximations plus expensive targets (Kandasamy et al., 2016).

These papers matter mostly as contrast, and that contrast is healthy. Our overnight controller used **none** of these ideas. It did not use GP surrogates, acquisition functions, multi-bracket early stopping, asynchronous promotions, or partial-run reuse. Every experiment was run to a full bounded horizon, then kept or discarded. That makes our controller simple and auditable, but it also means the paper should never pretend it discovered a better search strategy than the mature HPO literature.

2.5. 5 Automated ML and agentic research systems

Population Based Training sits near the boundary between HPO and adaptive training control. It demonstrates that hyperparameters can be changed online while training progresses, rather than chosen once and treated as fixed (Jaderberg et al., 2017). Auto-Sklearn 2.0 moves in a different direction: a more practical AutoML system that combines meta-learning, bandit-style budget control, and automated pipeline search under time limits (Feurer et al., 2020). Both are useful to cite because they show what more engineered automation looks like when the goal is not just one overnight result but a reusable system.

AutoML-Zero and The AI Scientist frame the more ambitious end of the automation spectrum. AutoML-Zero pushes toward open-ended algorithm discovery from primitive operations (Real et al., 2020), while The AI Scientist treats the entire research loop itself as an agentic target (Lu et al., 2024). Our work is deliberately smaller than either. That smallness is a feature, not a failure: one file, one metric, one benchmark, one machine, one ledger. The paper is stronger when it says this plainly.

3. Experimental Setup and Search Procedure

All experiments were run on a single NVIDIA GeForce RTX 3080 with 10 GB of VRAM using the Windows/RTX fork of `autoresearch` (`jsegov/autoresearch-win-rtx`). The runtime path used the PyTorch SDPA backend with `torch.compile` disabled, and the training surface was deliberately restricted to edits in `train.py`. This is worth stressing because it sharply limits what the autonomous loop was allowed to optimize: it could change training configuration and lightweight architectural knobs, but not the entire training stack.

The primary benchmark was TinyStories GPT-4 clean, and the metric of record was validation bits per byte (val_bpb), where lower is better. Each candidate received a fixed 300-second training budget plus startup and evaluation overhead. This fixed-horizon design is conceptually important. It means the search target was not "best asymptotic recipe" but "best use of five minutes on this card."

The search controller was host-side and sequential. It edited `train.py`, ran one experiment, parsed the reported metrics, appended the result to `results.tsv`, and advanced only when the new candidate improved on the best known val_bpb. Over the course of the campaign, 35 completed runs were accumulated, with 12 kept and 23 discarded. Once execution was moved to the host-side runtime rather than a mismatched delegated environment, the loop ran without crashes.

To test whether the overnight results were merely lineage luck, we then ran a separate two-hour confirmation sweep covering 17 additional runs across six selected configurations: the original baseline, the attention-only change, the best batch-size setting, the best depth setting, the near-best optimizer variant, and the final recommended recipe. Each configuration was rerun across 2-4 seeds, and we recorded mean val_bpb, VRAM, wall-clock time, and optimizer steps. This second pass was not part of the autonomous search controller; it was a post-search validation stage intended to distinguish stable effects from pretty accidents.

Just as important is what the controller did **not** do. It was not Bayesian optimization (Snoek et al., 2012), not Hyperband (Li et al., 2016), not ASHA (Li et al., 2018), not BOHB (Falkner et al., 2018), and not PBT (Jaderberg et al., 2017). It used no surrogate model, no early stopping brackets, no asynchronous promotions, no partial-run reuse, and no online population dynamics. That weakness is also its virtue: the resulting search trace is unusually easy to audit.

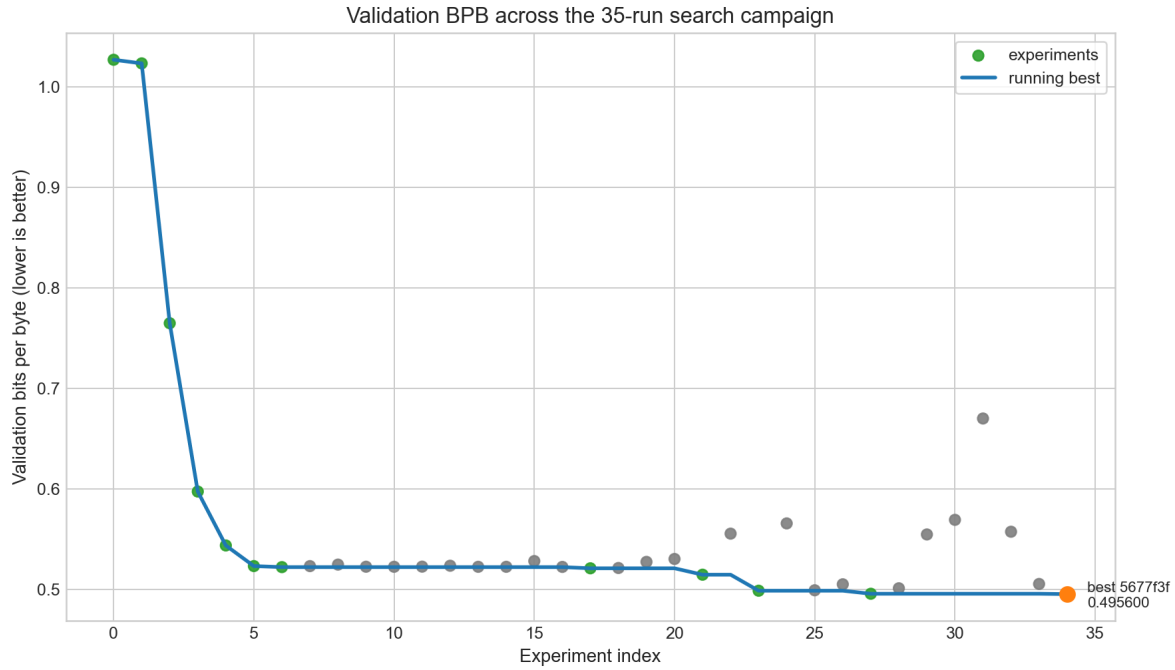

Figure 1: Validation BPB across the 35-run search campaign.

4. Main Results

The headline result has two layers. In the overnight search itself, the system moved from a baseline of 1.027091 val_bpb to a best single-run result of 0.495600, a 51.7% improvement over the original baseline and a 35.2% improvement over the pre-overnight starting point. In the follow-up confirmation sweep, the final recommended configuration averaged 0.497631 against a confirmed baseline mean of 0.996635, reproducing the main effect with a 50.1% improvement on mean performance. Figure 1 shows the original 35-run trajectory, and the curve has a clear shape: large early gains from compute-fit decisions, then smaller late gains from optimizer tuning.

The first apparent architectural result was the attention-pattern change. In the overnight search lineage, moving from the default mixed `SSSL` window pattern to `L` – full attention at every layer – produced a modest single-run improvement. The follow-up confirmation sweep, however, weakened that story substantially: the `window-L` configuration averaged 0.994334 versus 0.996635 for the confirmed baseline, a difference small enough that we should treat the standalone attention effect as inconclusive rather than settled. This is where the Longformer and BigBird citations still matter: local and sparse attention are serious efficiency ideas, which is exactly why a weak and unstable result should be described honestly instead of promoted into a fake theorem.

The dominant improvement axis, however, was batch size. Once total batch size was reduced from 2^18 to 2^17, 2^16, and finally 2^15, the score improved monotonically and dramatically. That is the cleanest empirical story in the whole run. It fits naturally with Smith et al. (2017): under a strict time cap, the number and usefulness of updates mattered more than preserving a large effective batch. Figure 2 captures that staircase clearly.

Depth was the second major lever. Reducing depth from 8 to 7 helped, and reducing it further to 6 helped even more, while increasing depth to 9 or 10 was actively harmful. That result matters because it says the five-minute regime is not rewarding latent capacity. It is rewarding architectures whose capacity can actually be exercised inside the compute horizon. Figure 3 shows that the shallower model was not just cheaper, but decisively better in this local regime.

Once batch size and depth had been brought into alignment with the budget, optimizer tuning generated the final gains. Pure schedule changes were modest: changing final LR fraction helped, while warmup/warmdown adjustments were mostly flat or worse. The last meaningful gains came from combined optimizer pressure – in particular, the move to `MATRIX_LR = 0.05`, `WEIGHT_DECAY = 0.1`, and eventually `EMBEDDING_LR = 0.8` on top of the shallower, smaller-batch configuration. The confirmation sweep suggests those optimizer changes were real but small: the final recipe averaged 0.497631, only

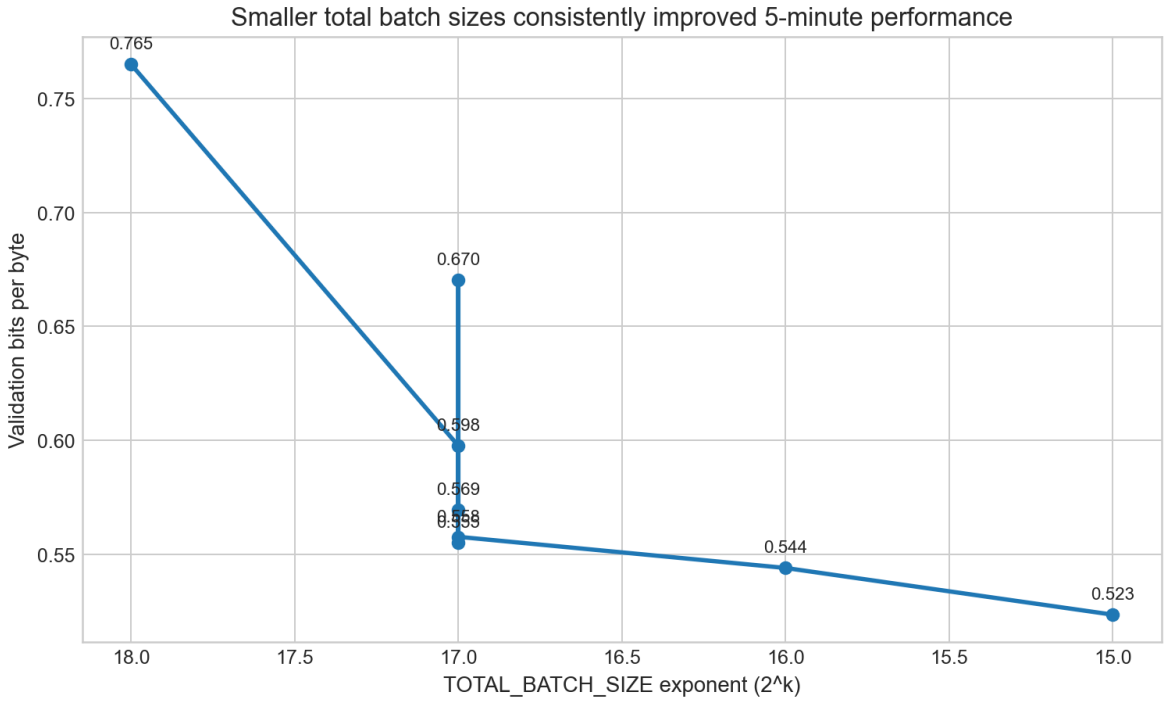

Figure 2: Smaller total batch sizes consistently improved 5-minute performance.

narrowly ahead of the depth-6 configuration at 0.498143 and the `MATRIX_LR = 0.05`, `WEIGHT_DECAY = 0.1` variant at 0.499665.

Memory behavior reinforces the same interpretation. The best results clustered around roughly 4.7-5.1 GB peak VRAM rather than near the card limit, which strongly suggests the loop was winning through better budget fit and update cadence rather than brute-force memory consumption. This was not a story of saturating memory harder; it was a story of using the available compute budget more effectively.

4.1. Post-search confirmation sweep

The follow-up confirmation pass is the most important credibility check in the paper because it separates regime shifts from single-lineage luck. Across 17 additional runs, the compute-fit story reproduced cleanly while the attention-only story weakened.

Label	Commit(s)	Runs	Mean val_bpb	Best val_bpb	Mean VRAM GB	Mean total sec	Mean steps	Interpretation
baseline-sssl	`a4123c6`	3	0.996635	0.916391	5.275	534.6	40.3	Confirmed reference point; high variance, slow, and clearly budget-misaligned
window-L	`8c6bcfe`	2	0.994334	0.924977	5.275	537.2	40.0	Standalone attention change was weak and inconclusive in follow-up

Continued on next page

Label	Commit(s)	Runs	Mean val_bpb	Best val_bpb	Mean VRAM GB	Mean total sec	Mean steps	Interpretation
batch-2e15	`3c0140f`	2	0.522191	0.522068	5.127	425.9	456.5	Strongly reproduced; smaller batch delivered many more useful updates
depth-6	`9637ff6`	3	0.498143	0.497026	4.654	385.4	788.7	Strongly reproduced; shallower model fit the five-minute budget much better
matrixlr005-wd01	`010edca`	3	0.499665	0.497731	4.654	385.9	790.7	Near-best optimizer variant; improvement remained incremental rather than structural
best-final	`5677f3f`	4	0.497631	0.496121	4.654	385.8	788.8	Best confirmed configuration on mean, but only by a narrow margin over depth-6

5. Best Configuration

The best experiment (`5677f3f`) used the following settings: `WINDOW_PATTERN = "L"`, `TOTAL_BATCH_SIZE = 2**15`, `DEPTH = 6`, `ASPECT_RATIO = 64`, `EMBEDDING_LR = 0.8`, `MATRIX_LR = 0.05`, `WEIGHT_DECAY = 0.1`, and `FINAL_LR_FRAC = 0.05`. This configuration should be read as the endpoint of a sequence, not as a lucky static guess. The search first solved the gross compute-fit problem (batch size and depth most clearly, with attention less securely), and only then extracted the remaining gains with optimizer tuning. The confirmation sweep matters here: `5677f3f` stayed best on mean across four seeds, but only by a hair over the simpler depth-6 recipe, which means the optimizer refinements should be described as marginal polish rather than a fresh regime change.

Figure 6 shows the top kept experiments. What is striking is how coherent the best run family looks: once the model was shallower and the batch smaller, several nearby optimizer combinations became competitive, but the final best configuration still won by a narrow margin. That suggests the search did not stumble into a single freak point so much as enter a better basin and then refine within it.

5.1. Table: Experiment family summary

Family	Knobs changed	Best commit	Best val_bpb	Interpretation
Baseline	Windows/RTX fork defaults (`SSSL`, large batch, depth 8)	`a4123c6`	1.027091	Starting point; clearly not budget-fit for this 3080 regime

Continued on next page

Family	Knobs changed	Best commit	Best val_bpb	Interpretation
Attention pattern	`WINDOW_PATTERN: SSSL -> L`	`8c6bcfe`	1.023505	Overnight single-run win, but the follow-up mean gain over SSSL was only 0.23%; standalone effect inconclusive
Batch-size sweep	`TOTAL_BATCH_SIZE: 2^18 -> 2^15`	`3c0140f`	0.523484	Dominant and strongly reproduced improvement axis; more useful updates beat larger batches
Schedule tuning	`FINAL_LR_FRAC`, warmup, warmdown	`99da6f5`	0.521284	Mild gains available, but smaller than batch/depth effects
Depth sweep	`DEPTH: 8 -> 7 -> 6`	`9637ff6`	0.498980	Strongly reproduced; shallower models fit the five-minute budget much better
Width sweep	`ASPECT_RATIO: 64 -> 48 or 80`	`4af2956`	0.499415 (discard)	Width changes were modest and did not beat the best depth fit
Optimizer-combo tuning	`MATRIX_LR`, `EMBEDDING_LR`, `WEIGHT_DECAY`	`5677f3f`	0.495600	Best single run and best confirmed mean, but only a narrow edge over depth-6

5.2. Top five kept runs

Commit	val_bpb	Peak VRAM (GB)	Status	Description
`5677f3f`	0.495600	4.7	keep	increase EMBEDDING_LR to 0.8 and MATRIX_LR to 0.05
`010edca`	0.495962	4.7	keep	increase MATRIX_LR to 0.05 and reduce WEIGHT_DECAY to 0.1
`9637ff6`	0.498980	4.7	keep	reduce DEPTH to 6
`87bc24f`	0.514914	5.0	keep	reduce DEPTH to 7
`99da6f5`	0.521284	5.1	keep	set FINAL_LR_FRAC to 0.05

6. Discussion

The cleanest interpretation of the run set is that it discovered **compute fit** before it discovered anything else. The major gains that survived confirmation came from reducing total batch size and reducing depth. This is exactly the type of local budget-shaped behavior that scaling-law and compute-optimal work would lead us to expect in spirit, even if our setup is far too small and narrow to claim their formal laws directly (Kaplan et al., 2020; Hoffmann et al., 2022).

The attention result is now more interesting precisely because it got weaker. Sparse and local attention exist for good reasons (Beltagy et al., 2020; Zaheer et al., 2020), but those reasons are not automatically binding in every low-budget regime. Our overnight lineage suggested that plain full attention beat the mixed-window setting. The follow-up confirmation sweep, however, showed only a tiny mean advantage for `L` over `SSSL`, which means the honest reading is that the attention-only effect remains unresolved in this regime. A plausible explanation is that any true difference is small relative to seed noise at this budget horizon.

The optimizer story is subtler. The batch-size result matches the idea that update cadence can dominate under strict time caps (Smith et al., 2017). The late-stage optimizer gains suggest a second

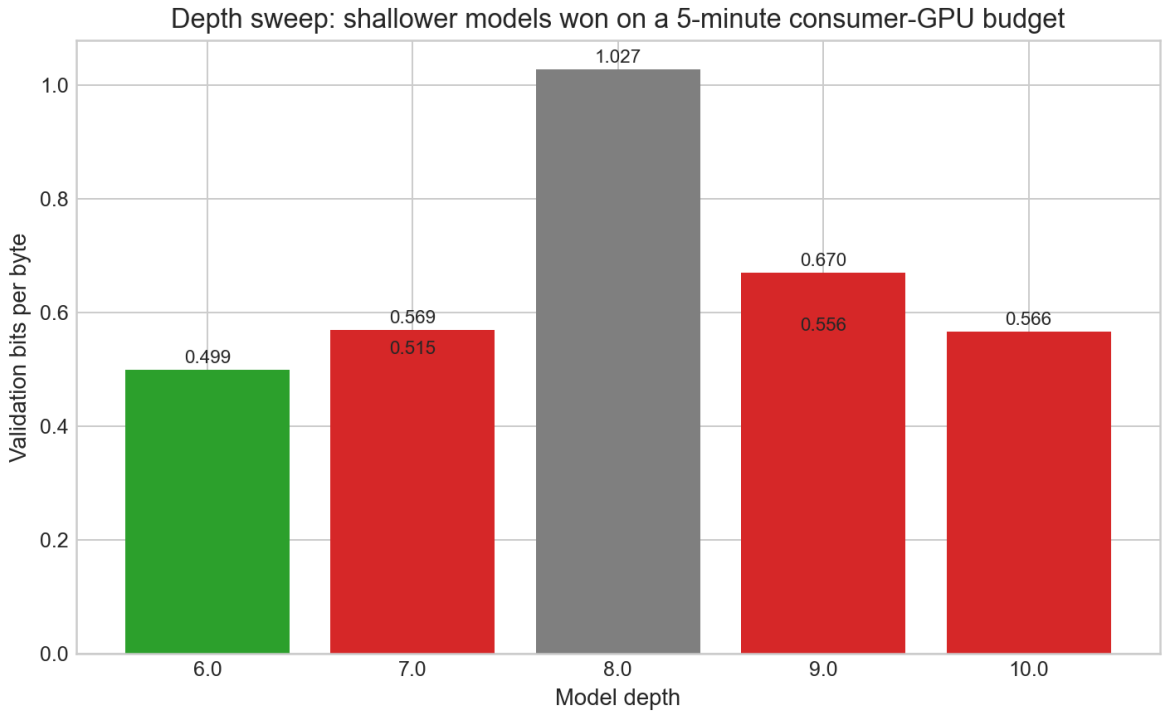

Figure 3: Depth sweep: shallower models won on a 5-minute consumer-GPU budget.

layer: once the architecture and batch size were made budget-compatible, the search could finally benefit from stronger learning pressure and better regularization. The confirmation sweep supports this, but only weakly: the best-final recipe stayed on top, yet only narrowly. That means the optimizer refinements look like local polish inside a better basin, not a giant second breakthrough.

One deeper lesson hides in the step counts. The baseline family consumed roughly 535 seconds for only about 40 optimizer steps, whereas the confirmed depth-6 and best-final recipes used roughly 386 seconds to deliver about 789 steps. That is not a cosmetic speedup; it is a phase change in how the card is being used. Much of what looks like "model quality" on a consumer GPU is really whether the wall-clock budget is long enough for learning to begin in earnest.

A second implication is that excess capacity under a hard wall-clock budget behaves less like latent potential and more like dead weight. The deeper models were not merely a little worse; they spent scarce wall-clock and memory without translating that cost into useful parameter movement. For small labs, that suggests a harsher but cleaner discipline: first solve for update-efficient regimes, then spend whatever budget remains on expressivity. Otherwise, bigger models become a status purchase paid in lost experiments.

A third implication is epistemic. Once batch size and depth were fixed, the remaining contenders clustered tightly around 0.498-0.500 val_bpb. That narrowing matters because it tells us when the search has moved from first-order discovery to second-order polish. On a card like the 3080, the practical skill is not just finding a better point; it is knowing when the frontier has flattened enough that another night of scalar fiddling is less valuable than testing a new mechanism class.

Finally, the search-controller story is both the weakest and most interesting part of the paper. By the standards of BO, Hyperband, ASHA, BOHB, Freeze-Thaw BO, or Fabolas, our controller is primitive (Snoek et al., 2012; Li et al., 2016; Li et al., 2018; Falkner et al., 2018; Swersky et al., 2014; Klein et al., 2016). But that primitiveness is also why the result is believable. A narrow, auditable, one-lineage keep/discard loop uncovered a 51.7% single-run improvement over baseline, and the follow-up confirmation sweep reproduced the main effect with a 50.1% improvement on mean performance. The implication is not that we invented a better HPO method. The implication is that constrained agentic search can already be useful before it becomes sophisticated.

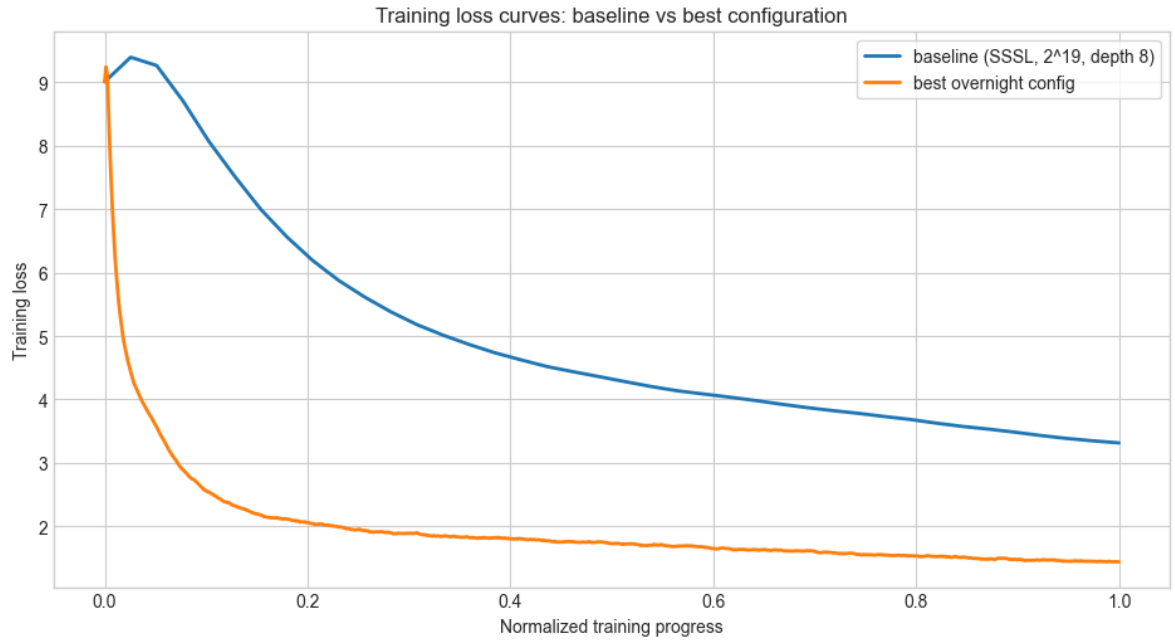

Figure 4: Training loss curves: baseline versus best configuration.

7. Limitations

This report remains highly local. The result is tied to one dataset, one hardware envelope, one runtime fork, one metric, and one narrow search surface. TinyStories is intentionally narrow and low-entropy (Eldan and Li, 2023), so there is no reason to assume that the same configuration would remain optimal on broader corpora or longer training schedules. Similarly, the behavior of `WINDOW_PATTERN`, batch size, and depth is almost certainly regime-dependent.

The confirmation sweep improves the paper, but it does not magically make it broad. It covered only six selected configurations, only 2-4 seeds per configuration, and only the same 3080/runtime path. That is enough to test whether the main story survives contact with repeated runs, but not enough to map the full stability landscape or to establish precise effect sizes for small deltas such as the attention-only change.

The controller is also intentionally weak. We did not benchmark against Practical BO, Hyperband, ASHA, BOHB, Freeze-Thaw BO, Fabolas, or multi-fidelity GP-bandit methods. That means the paper offers no evidence that our search process is competitive with mature HPO systems. At most, it shows that a very simple loop can still find valuable local improvements when the problem is tightly constrained.

There are also important unexplored neighboring mechanisms. We did not test compression or distillation strategies such as DistilBERT, did not test optimizer geometry methods such as SAM, and did not compare directly against dedicated sparse-attention families such as Longformer or BigBird. Those omissions should be read as future-work openings, not as hidden wins.

In short, the right claim boundary is narrow: this paper is a strong empirical note about a fixed five-minute consumer-GPU training regime, plus a small post-search confirmation pass, not a general theorem about language-model scaling, attention design, or autonomous science.

8. Practical Future Work on a 3080

The most actionable conclusion from this paper is that a 3080 becomes useful when treated as a high-iteration empirical instrument rather than a miniature frontier-training cluster. Its comparative advantage is not raw model scale. It is the ability to run enough bounded experiments cheaply enough that hardware-specific training laws can be discovered instead of guessed.

First, the current loop should be upgraded with budget-aware search rather than more blind lineage exploration. Hyperband, ASHA, BOHB, Freeze-Thaw BO, and Fabolas all point toward the same operational move: use partial curves, pauses and resumes, and cheap fidelities to reallocate compute toward promising candidates. Given that the baseline used roughly 535 seconds for about 40 steps while

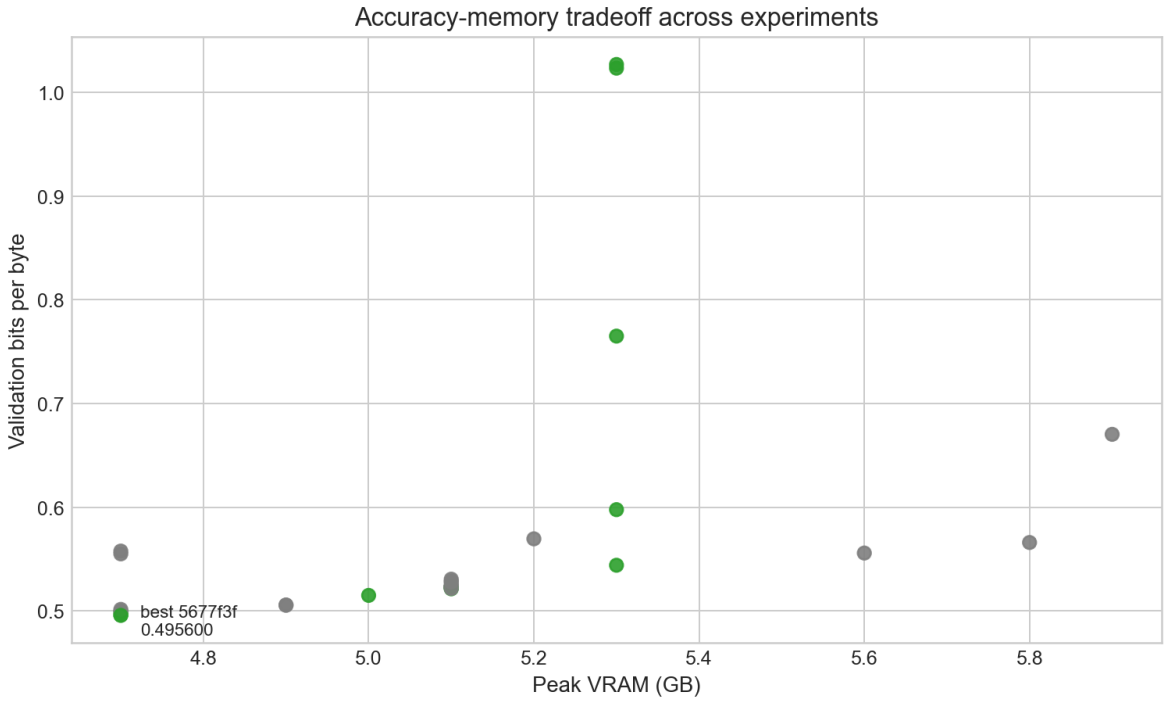

Figure 5: Accuracy-memory tradeoff across experiments.

the confirmed best family used roughly 386 seconds for about 789, the card is practically begging for a controller that reasons about useful updates per wall-clock second rather than treating every full run as equally informative.

Second, the 3080 is a strong platform for a two-stage discover-then-distill workflow. The role of the overnight search is to find a compute-fit teacher recipe and a stable basin cheaply. The role of a follow-on stage, inspired by DistilBERT, would be to compress or transfer that behavior into a smaller deployment artifact or a longer downstream training trajectory. In that framing, the 3080 is not just a trainer; it is a recipe-discovery engine.

Third, the next optimization frontier likely lies in quality per step, not just more steps. Once batch size and depth were fixed, the remaining gains were narrow and optimizer-shaped. That is exactly the zone where methods like SAM, better schedule adaptation, or online hyperparameter adjustment via PBT could matter. The right question is no longer "how do we fit more model on the card?" but "which mechanisms buy more generalization from each scarce update?"

Fourth, the attention question should be revisited only under a design that can actually resolve it: more seeds, longer contexts, and perhaps a direct comparison against dedicated sparse-attention families rather than only a pattern toggle inside the same codepath. Right now the honest lesson is not that sparse attention failed. It is that under a five-minute 3080 budget, fancy attention did not obviously pay before compute fit was solved.

In short, the way to make a 3080 useful is to make it ruthlessly local, iterative, and auditable. Let bigger systems chase asymptotic scale. Let the 3080 discover which ideas are real before you spend serious compute on them.

9. Conclusion

A host-side autonomous search loop improved a small language-model training recipe on commodity hardware from 1.027091 to 0.495600 val_bpb in the overnight search, and a 17-run follow-up confirmation sweep reproduced the main effect with a best-final mean of 0.497631 against a confirmed baseline mean of 0.996635. The path to that result was straightforward but revealing: shrink total batch size, shrink depth, and then retune optimization once the model finally fit the compute budget. The attention-only claim, by contrast, weakened in follow-up and should be treated as unresolved. That mix – one story getting stronger, another getting softer – is exactly why the result is interesting. It is legible, more reproducible than a single-run anecdote, and grounded in the actual machine rather than in aesthetic

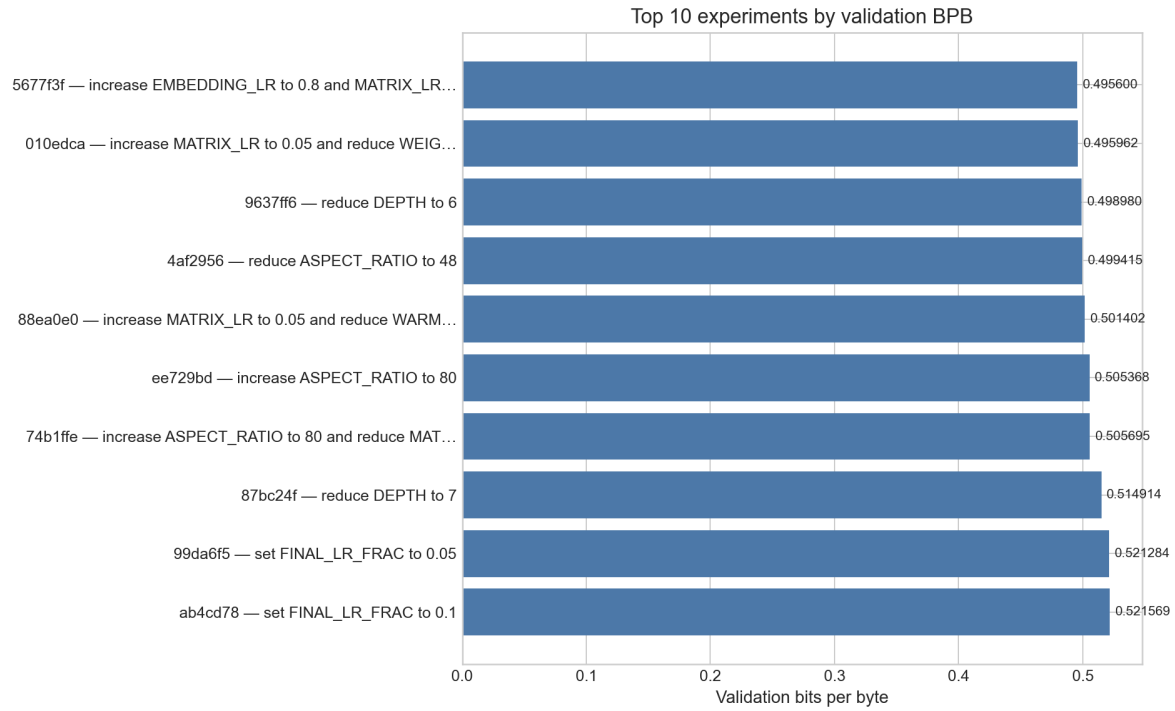

Figure 6: Top kept experiments by validation BPB.

beliefs about what should be optimal.

The broader lesson is modest but useful. Constrained agentic research does not need to start by solving research automation in the large. It can start by solving a small, bounded, measurable problem with a clean ledger, then run a second pass to see which claims survive. If anything, this report suggests that narrow search spaces may be the most productive habitat for autonomous research systems before they move into broader and harder-to-audit regimes.

A. Literature Positioning Table

Paper	Problem class	Search / mechanism type	Resource model	Relation to our work
Attention Is All You Need	Transformer architecture	Full self-attention baseline	Standard training	Foundation for full-attention comparison
Longformer	Long-context transformers	Local + global windowed attention	Linearized long-sequence efficiency	Contrast for mixed-window patterns
Big Bird	Sparse-attention transformers	Sparse/global attention	Long-sequence efficiency	Contrast for sparse-attention design
DistilBERT	Compact transformer models	Distillation / compression	Smaller faster student models	Contrast for small-model efficiency beyond tuning
SAM	Optimizer / generalization	Sharpness-aware optimization	Better generalization per step	Mechanism candidate beyond hyperparameter tuning
Scaling Laws	LM scaling	Power-law empirical scaling	Compute-aware training	Budget-shaped framing
Chinchilla	Compute-optimal LMs	Compute/data allocation	Fixed-compute tradeoff	Strong conceptual framing for bounded budget
Don't Decay LR, Increase Batch Size	Optimization dynamics	Batch-size / schedule relationship	Update-cadence view	Mechanism for batch-size interpretation

Continued on next page

Paper	Problem class	Search / mechanism type	Resource model	Relation to our work
Practical BO	HPO	Gaussian-process Bayesian optimization	Expensive black-box search	Stronger HPO baseline class
Hyperband	HPO	Multi-bracket early stopping	Fixed budget allocation	Stronger budget-aware search contrast
ASHA	HPO	Asynchronous early stopping	Better wall-clock use	Stronger controller contrast
BOHB	HPO	Multi-fidelity + model-based search	Cost-aware guided search	Stronger model-based search contrast
Freeze-Thaw BO	HPO	Partial-run reuse / resume	Iterative training curves	Mechanism for smarter bounded loops
Fabolas	HPO	Dataset-fidelity BO	Cheap proxy evaluations	Mechanism for multi-fidelity improvement
MF-GP-UCB	HPO	Multi-fidelity GP bandit optimization	Cheap approximations + expensive target	Theory for fidelity-aware search
PBT	HPO / training	Online adaptation	Fixed compute with evolutionary replacement	Precedent for adaptive training control
Auto-Sklearn 2.0	AutoML systems	Meta-learning + budget-aware AutoML	Time-limited practical AutoML	System-level contrast
AutoML-Zero	Automated ML discovery	Open-ended evolutionary search	Generic search over primitive ops	Contrast with narrow search space
AI Scientist	Agentic research systems	End-to-end experiment + writing loop	General autonomous workflow	Framing contrast / precedent
TinyStories	Small-model benchmark	Controlled low-entropy dataset	Cheap language-model experimentation	Benchmark legitimacy

B. Reproducibility Table

Item	Value
Workspace root	`C:/openclawworkspace`
Paper workspace	`C:/openclawworkspace/papers/autoresearch-3080-overnight-2026-03-22`
Repo path	`C:/openclawworkspace/autoresearch`
Working branch	`autoresearch/mar21-3080`
Current / best commit	`5677f3f`
Overnight run window	`2026-03-21 23:01:14` to `2026-03-22 02:48:32`
Overnight run directory	`C:/openclawworkspace/autoresearch/overnight-runs/20260321-230114`
GPU	`NVIDIA GeForce RTX 3080`
VRAM	`10 GB`
Runtime fork	`jsegov/autoresearch-win-rtx`
Attention backend	`PyTorch SDPA`
Compile mode	`torch.compile disabled`
Dataset	`TinyStories GPT-4 clean`
Search controller	`host-side sequential keep/discard loop`
Search completed runs	`35`
Search kept / discarded / crashed	`12 / 23 / 0`
Confirmation run window	`2026-03-22 09:34:28` to `2026-03-22 11:38:45`
Confirmation run directory	`C:/openclawworkspace/autoresearch/confirmation-runs/20260322-093428`
Confirmation completed runs	`17`
Total completed runs in report	`52`
Search baseline val_bpb	`1.027091`
Search best val_bpb	`0.495600`
Confirmed baseline mean val_bpb	`0.996635`
Confirmed best-final mean val_bpb	`0.497631`
Primary ledger	`autoresearch/results.tsv`
Overnight summary	`autoresearch/overnight-summary.md`
Confirmation summary	`autoresearch/confirmation-runs/20260322-093428/confirmation-summary.md`
Host runner script	`scripts/autoresearch_overnight_runner.py`
BibTeX companion	`references.bib`

C. Artifact Manifest

- Main experiment ledger: `C:/openclawworkspace/autoresearch/results.tsv`
- Overnight narrative summary: `C:/openclawworkspace/autoresearch/overnight-summary.md`
- Overnight per-run logs: `C:/openclawworkspace/autoresearch/overnight-runs/20260321-230114/logs`
- Confirmation results ledger: `C:/openclawworkspace/autoresearch/confirmation-runs/20260322-093428/conf`
- Confirmation narrative summary: `C:/openclawworkspace/autoresearch/confirmation-runs/20260322-093428`
- Confirmation per-run logs: `C:/openclawworkspace/autoresearch/confirmation-runs/20260322-093428/logs`
- Paper drafts: `paper.md`, `paper_v2.md`, `paper_v3.md`, `paper_v4.md`, `paper_v5.md`, `paper_v6.md`, `paper_v7.md`
- Paper PDFs: `paper.pdf`, `paper_v2.pdf`, `paper_v3.pdf`, `paper_v4.pdf`, `paper_v5.pdf`, `paper_v6.pdf`, `paper_v7.pdf`
- Figures: `figures/*.png`
- Enriched data table: `data/results_enriched.csv`
- Reference packs: `references/arxiv-seed-5`, `references/arxiv-seed-10`, `references/arxiv-seed-20`
- BibTeX companion file: `references.bib`

References

[1] Ronen Eldan and Yuanzhi Li. *TinyStories: How Small Can Language Models Be and Still Speak Coherent English?* arXiv:2305.07759, 2023.

[2] Ashish Vaswani et al. *Attention Is All You Need.* arXiv:1706.03762, 2017.

[3] Jared Kaplan et al. *Scaling Laws for Neural Language Models.* arXiv:2001.08361, 2020.

[4] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models.* arXiv:2203.15556, 2022.

[5] Samuel L. Smith et al. *Don't Decay the Learning Rate, Increase the Batch Size.* arXiv:1711.00489, 2017.

[6] Pierre Foret et al. *Sharpness-Aware Minimization for Efficiently Improving Generalization.* arXiv:2010.01412, 2020.

[7] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* arXiv:1910.01108, 2019.

[8] Iz Beltagy et al. *Longformer: The Long-Document Transformer.* arXiv:2004.05150, 2020.

[9] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences.* arXiv:2007.14062, 2020.

[10] Max Jaderberg et al. *Population Based Training of Neural Networks.* arXiv:1711.09846, 2017.

[11] Jasper Snoek et al. *Practical Bayesian Optimization of Machine Learning Algorithms.* arXiv:1206.2944, 2012.

[12] Kevin Swersky et al. *Freeze-Thaw Bayesian Optimization.* arXiv:1406.3896, 2014.

[13] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization.* arXiv:1603.06560, 2016.

[14] Kirthevasan Kandasamy et al. *Multi-fidelity Gaussian Process Bandit Optimisation.* arXiv:1603.06288, 2016.

[15] Aaron Klein et al. *Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets.* arXiv:1605.07079, 2016.

[16] Liam Li et al. *A System for Massively Parallel Hyperparameter Tuning.* arXiv:1810.05934, 2018.

[17] Stefan Falkner et al. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale.* arXiv:1807.01774, 2018.

[18] Matthias Feurer et al. *Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning.* arXiv:2007.04074, 2020.

[19] Esteban Real et al. *AutoML-Zero: Evolving Machine Learning Algorithms From Scratch.* arXiv:2003.03384, 2020.

[20] Chris Lu et al. *The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery.* arXiv:2408.06292, 2024.

[21] Andrej Karpathy. *autoresearch.* GitHub repository, 2026.

[22] J. Segov. *autoresearch-win-rtx.* GitHub repository, 2026.

[23] Andrej Karpathy. *nanochat.* GitHub repository, 2026.

[24] TinyStories GPT-4 clean dataset. Hugging Face dataset card, 2026.