Most of a Formal Proof is Forced

Po Hung Yeh [1] **Luke Ong** [1]

Abstract

A language model that writes mathematics can produce something that looks like a proof and is not one. We grade against the Lean 4 lambda calculus kernel: a candidate is accepted when its inferred type matches the stated goal, using a binary metric that cannot be hallucinated. On this surface, the next token is largely forced by the goal type and elaboration context, with the residual entropy concentrated at a sparse set of genuine decision points. A from-scratch and transferred scaling sweep reads loss exponents three to four times steeper than the natural-language reference on both data and parameter axes, with a small irreducible floor; the exponents are calibration-invariant to the tokenisation scheme. For models larger than 30M, model size becomes redundant, and loss is set by effective data alone. The concentrated entropy makes depth-first search cheap: one model call per emitted step boundary, free chain rollout between them. We demonstrate nontrivial empirical performance with supervised fine-tuning alone, paving the way for a potentially competitive prover.

1. Introduction

A language model that writes prose is wrong by being unhelpful. A language model that writes mathematics is wrong in a stronger way: past work generates proof objects via control of the surface syntax tactic strings (Yang et al., 2023; Xin et al., 2024) and relies on Lean's elaborator to translate that into the `Lean.Expr` which the kernel then verifies. The elaborator is the compiler between the training signal and the kernel grade. In this paper, we remove the layer by directly generating in `Lean.Expr`: the tree of constructors Lean 4's kernel type checks (de Moura & Ullrich, 2021), a (representation of) dependently typed lambda calculus in all but name. Every term our model generates goes back to the kernel. There is no learned judge (Lightman et al., 2024), no proof-state reformulation between model calls (Yang et al., 2023). We call the resulting apparatus the *Church Machine*: a model that proposes and reasons in lambda terms and Lean's kernel as the only metric. A Church Machine inherits Lean's defining advantage: it can never hallucinate a theorem. It can produce a tree that does not parse, one that parses but does not type-check, one that type-checks but proves something other than the goal, yet, inherited from Lean, it cannot produce a kernel-accepted term that is nonetheless wrong.

The metric. We are given a held-out collection of (type, value) pairs $\mathcal{T} = \{(t_i, e_i)\}_{i=1}^{n}$ where t_i is a goal and e_i a Lean term the kernel already accepts as inhabiting t_i, and a generator g_θ producing, for a type t, an ordered list of candidates $g_\theta(t)_{(1)}, \ldots, g_\theta(t)_{(k)}$. The *Kernel Acceptance Rate* (KAR) at budget k is

$$\begin{aligned}\mathrm{KAR}_k(g_\theta; \mathcal{T}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\big[\, \exists\, b \le k : \\ \texttt{Lean.Meta.check}\big(g_\theta(t_i)_{(b)}\big) \wedge \\ \texttt{isDefEq}\big(\mathrm{inferType}(g_\theta(t_i)_{(b)}),\, t_i\big)\,\big]. \quad (1)\end{aligned}$$

A candidate c is accepted just if it type-checks and the kernel's inferred type for it is definitionally equal to the stated goal. Note that the check is on the type, not the term: we never ask whether a candidate c equals the held-out witness e_i, only whether c inhabits t_i. For t_i : Prop there is no gap to exploit: by proof irrelevance every inhabitant of a proposition is definitionally equal to every other, so once the kernel confirms that c inhabits t_i, c is automatically definitionally equal to e_i. [1] For t_i : Type the leniency is real: any kernel-accepted inhabitant of t_i counts. The metric is deliberately lenient, yet a lenient metric is gameable: against a weak goal type a *degenerate* term will still pass. Nevertheless, the metric is exact by asking the question we want: did the generator produce a term that the kernel agrees inhabits the stated goal?

A failed candidate falls into one of three buckets we keep apart throughout: parse_err (the token stream does not de-

[1]Nanyang Technological University. Correspondence to: Luke Ong <ntu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026. Copyright 2026 by the author(s).

[1]We have employed an axiom screen to prevent the model from bypassing the theorem using a custom axiom.

code into a well-formed `Lean.Expr`), type_err (it decodes but the kernel rejects it), neq (it type-checks but the inferred type disagrees with t_i). These are qualitatively different problems, and the contrast between their populations carries representation signal we use in Section 3.1.

A change of substrate. The Church Machine removes the prose layer: it generates `Lean.Expr` directly, so the reasoning medium *is* the proof object: application, abstraction, and the type of each sub-term constitute the logical structure rather than describing it. The model is handed no natural-language scaffold; it infers the context from the typed term alone. Hence we name it after Alonzo Church, whose Lambda Calculus we take as the reasoning language itself. The novelty is not a higher benchmark score but *a change of substrate*: the proof object stops being the downstream artefact of natural-language reasoning and becomes the reasoning trace itself. Evidence for the value of the change of substrate lies in the shape of the scaling law of the Church Machine (Section 3.1): both parameter and data exponents are three to four times steeper than the natural-language reference, and the residual entropy concentrates at branch points which proof search can exploit (Section 3.3).

2. Methodology

2.1. Prerequisites

Lean expression and elaboration When discussing Lean code, one typically refers to its *surface syntax*. Under the hood, Lean translates the surface syntax into a canonical form `Lean.Expr`[2] which is the term language of the Lean kernel. This translation is called *elaboration*. Precisely `Lean.Expr` is (the internal AST of) a dependently typed lambda calculus, enriched with constants, metavariables, and elaboration machinery used by the Lean kernel to represent terms, types, proofs, definitions and propositions. The Church Machine operates at this lambda-calculus layer. For a comprehensive online Lean reference, we recommend https://leanprover-community.github.io/learn.html.

Scaling laws studies Before committing GPU-months to a single methodological bet, we first measure the *scaling laws* of the Church Machine: empirical functional forms, such as (2), that predict test cross-entropy loss L, given model parameter count N, training dataset size D (in token numbers), and compute budget C (in FLOP). Small-scale fits extrapolate reliably over several orders of magnitude (Kaplan et al., 2020; Hoffmann et al., 2022), so the sweet-spot allocation can be identified before any large run begins. Scaling laws are therefore the first steps in our investigation: they bound the loss reachable for a given budget, and so can tell us whether a methodological gain is the binding constraint.

Chinchilla scaling law Hoffmann et al. (2022), based on earlier work of Kaplan et al. (2020), fit the functional form

$$L(N, D) \;=\; E \;+\; A \cdot N^{-\alpha} \;+\; B \cdot D^{-\beta} \tag{2}$$

where $E \approx 1.69$ nats is the irreducible entropy of natural texts, $\alpha \approx 0.34$, and $\beta \approx 0.28$.

Data-constrained scaling When the available corpus is smaller than the compute-optimal $D^\star$, additional compute can only be spent by repeating data; Muennighoff et al. (2023) showed that repeated tokens are worth less than fresh ones, with returns indistinguishable from zero beyond roughly four epochs. Equation (2) therefore serves as a lower bound for the data-constrained regime that our formal-mathematics corpus inhabits.

Transfer scaling law Pre-training amounts to *effective* fine-tuning data. Hernandez et al. (2021) made this precise: in the low-data regime, the additional data D_T that a from-scratch model[3] of the same size would need to match a pre-trained model's loss follows

$$D_T \;=\; k\, D^{\,\alpha_T}\, N^{\,\beta_T} \tag{3}$$

where D is the fine-tuning set size, N is the parameter count, and (α_T, β_T) are transfer exponents distinct from those of (2). Because we generally cannot determine whether an off-the-shelf pretrained model with weights w is under fixed experimental control, we absorb the effective data as $D + D_T = m_w \cdot D$ and fit m_w per warm-start class. The updated scaling fits

$$L(N, D) \;\approx\; E \;+\; A \cdot N^{-\alpha} \;+\; B \cdot \big(m_w \cdot D\big)^{-\beta} \tag{4}$$

Our empirical dominant term is $B \cdot \big(m_w \cdot D\big)^{-\beta}$, indicating that data size is our bottleneck.

2.2. Corpus and decomposition

What the corpus contains. The corpus is drawn from eight Lean 4 libraries (Table 1): Mathlib at its development head, the language core (`Init`, `Std`), the Lean compiler's own metaprogramming internals, the Batteries and Aesop libraries, and two public collections of formalised competition problems (Lean Workbook (Ying et al., 2024) proof generated by Lin et al. (2025) and NuminaMath-Lean (Liu et al.,

[2]The term language `Lean.Expr` is based on a dependent type theory with universes, inductive types, local binders, reducibility annotations, and metavariables, in the tradition of Martin-Löf Type Theory (Martin-Löf, 1998) and the Calculus of Inductive Constructions (Coquand & Huet, 1988).

[3]meaning training a model entirely from zero, starting with completely random weights and no prior exposure to language

Corpus	Decls.	With value
Mathlib (HEAD)	293,144	53,726
Lean `Init`	32,205	7,327
Lean `Std`	27,159	6,514
Lean internals	21,906	20,799
Batteries	3,399	1,392
Aesop	2,409	2,293
Lean Workbook	5,725	1,964
NuminaMath–Lean	2,375	1,689
Total	388,322	95,704

Table 1. The eight source libraries. *Decls.* counts non-internal declarations extracted; *With value* counts those carrying a proof term or definition body after oversize terms are value-capped. The compiler internals are almost all short definitions and keep nearly every value; The value-bearing column is the training pool.

2026)), filtered by compilation on the target toolchain. The extraction is 388,322 non-internal declarations, of which 95,704 carry a value. Of these, 92,051 are definition bodies and only 3,653, all from the two competition sets, are theorem proofs: Lean's kernel stores a proved theorem as its type alone, tagged inhabited, and discards the proof term, so the values we recover from the six compiled libraries are definition bodies, not theorem proofs[4]; and those value-bearing rows are what the model trains on.

We keep proofs and programs in one corpus by design: under the Curry–Howard Correspondence (Curry, 1934; Howard, 1980) they are the same object, i.e. a typed term the kernel checks. Thus, the compiler's functional-programming code is included alongside the theorem proofs Since the kernel-grounded surface does not separate the two, we believe the model should not learn a boundary that is not there.

We study three properties of the kernel-grounded surface:

1. entropy of its conditional distribution under scaling
2. data-axis of transfer from natural language, and
3. proof search algorithms the entropy shape admits.

Section 3 presents three falsifiable claims and the measurements that support them. The remainder of this subsection describes how a Lean proof is decomposed into the sequence on which the language model trains, making the claims measurable.

From elaboration tree to token stream. Given a value-bearing `Lean.Expr` tree, we transform it to a string by a pre-order, depth-first walk — root before children,

[4]We overlooked that the kernel discards the proof value during elaboration when we intended to include it. It will be fixed in future work.

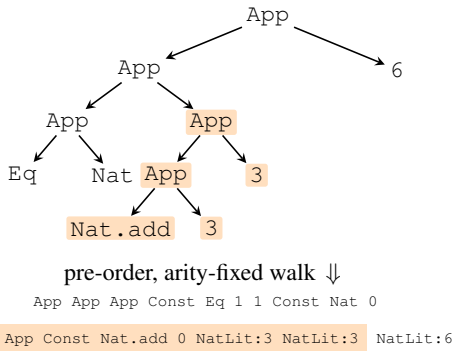

Figure 1. **Natural serialisation of a `Lean.Expr` tree.** The elaboration tree for $3+3=6$ (the kernel-level `@Eq Nat (Nat.add 3 3) 6`) is linearised by a pre-order, depth-first walk, root before children. Every constructor has a fixed arity (`App` is always binary) so the token stream below the tree needs no brackets yet decodes back to the tree uniquely, and bound variables would appear as de Bruijn indices.

Elaboration tree	$\Longleftrightarrow$	Emitted token stream
(root) $a+0=a$: Prop	$\Longleftrightarrow$	`<bos><corpus_X>` $a+0{=}a$ `<sep_type>`
subterm **1** : Nat → . . .	$\Longleftrightarrow$	`<step_begin>. . . <step_end><accept>`
subterm **2** : Nat	$\Longleftrightarrow$	`<step_begin>. . . <step_end><accept>`
subterm **3** : BVar 0	$\Longleftrightarrow$	`<step_begin>. . . <step_end><accept>`
(commit boundary)	$\Longleftrightarrow$	`<commit>` *full witness term* `<eos>`

Figure 2. Decomposing a Mathlib declaration into the chain-of-thought stream the model trains on. Each closed subterm of the elaboration tree (left, listed root-to-leaf) is emitted as a `<step_begin>`...`<step_end>` block followed by a kernel verdict (`<accept>`, shown). The `<commit>` boundary separates the chain of thought from the full witness term the kernel grades at the end.

bound variables made locally nameless as de Bruijn indices, constructor arities fixed so the stream needs no brackets and decodes back to the tree exactly, as presented in Figure 1. We call this transformation the *natural serialisation*: the tree read in the order the kernel built it, with nothing added. Each closed subterm encountered becomes one *step* in the reasoning (Figure 2), with three pieces emitted: the subterm's expected type τ_{exp} at that elaboration position, the subterm itself, and a kernel verdict computed by re-running `Lean.Meta.check` followed by $\texttt{isDefEq}(\text{inferType}(\cdot),\ \tau_{\text{exp}})$ on the closed subterm in isolation. On about 5% of steps we inject a perturbed subterm, a parse-error or type-error variant, or a subterm tagged with the expected-type probe slot `<sep_probe_type>`, so the model sees the full verdict vocabulary at training time, not only accept. The verdict is one of accept, parse_err, type_err, neq. The model trains on a stream in which its own predicted tokens are interleaved with verdicts the kernel provided. The kernel is the runtime's observation; the model is never asked to certify itself.

Each row of the packed corpus thus has the schema

Unit (median)	English (Qwen3 tok.)	`Lean.Expr` (our tok.)	Ratio
Theorem statement	26	197	7.6×
Proof term	148	2014	13.6×

Table 2. Length calibration. On $N = 200$ competition theorems (LEAN WORKBOOK + NUMINA, sampled uniformly), the kernel `Lean.Expr` is *considerably more verbose*: a theorem statement costs $\sim 7.6\times$ #tokens of its plain-English rendering (median over the 200 rows), a proof term costs $\sim 14\times$ (likewise the median).

<bos> <corpus_X> $\{\tau\}$ <sep_type>
$\big[$ <step_begin> $[\{\tau_{\text{exp}}\}$ <sep_probe_type>$]$
$\{$sub_term$\}$ <step_end> $\langle$verdict$\rangle$ $\big]^*$
<commit> $\{$full value$\}$ <eos>

where <corpus_X> tags the source library (Mathlib, stdlib, etc.) and the bracketed expected-type slot is optional at each step. Loss is computed on every token except the position-0 <bos>; the verdict tokens (accept, parse_err, type_err, neq) are kernel-provided context and are masked out, so the model is never trained to predict them.

3. Lambda Calculus Scaling Law

3.1. The kernel-grounded surface is mostly forced

`Lean.Expr` is essentially a typed lambda calculus. Since it is such a low-level programming language, representing high-level programs in `Lean.Expr` results in massive combinatorial expansion. In this sense, the lambda calculus is a verbose language, costing $14\times$ as many tokens as the natural-language rendering (Table 2) in exchange for full kernel-checkability and, as we measure below, a per-token entropy floor far below the natural-language reference.

Claim. The kernel-grounded surface appears *mostly forced*: outside a sparse minority of decision points, the next token is well-predicted by the goal type and the elaboration context. Viewed through Shannon's lens—every model's cross-entropy is an upper bound on the true source entropy—the Lambda Calculus is a *near-minimum-entropy carrier* for mathematics, possibly approaching the irreducible information cost of the proof content itself.

Sweep. The from-scratch scaling law grid has twelve cells: four model sizes ($N \in \{6, 17, 38, 100\}$ M non-embedding parameters) by three token budgets ($D \in \{1, 4, 16\} \times D_{\text{unique}}$ where $D_{\text{unique}} \approx 95$ M tokens). All cells share the same architecture, the natural serialisation, and AdamW at cosine schedule with linear warm up. Training is single-GPU per cell, ~ 4096-token contexts with intra-document packing (Zhao et al., 2024) under a block-diagonal attention mask. The corpus is the kernel-verified probe-form pack of Section 2.2 derived from Mathlib plus its standard libraries and two competition-problem extracts.

Fit. Least squares in log-space against the unified loss form

$$L(N, D, w) \;=\; E \;+\; A \cdot N^{-\alpha} \;+\; B \cdot (m_w \cdot D)^{-\beta} \quad (5)$$

where m_w is a data multiplier applied for transfer learning (fixed at 1 for from-scratch and fitted for each Qwen3 body in Section 3.2) yields the coefficients of Table 3. Two readings stand out. First, the *exponents* α and β, which are invariant to length calibration (Section C), are $3\times$ to $4\times$ steeper than the natural-language reference: a doubling of either model size or token budget buys a loss reduction whose log-decade slope is $3\times$ to $4\times$ that of web text. The second reading is the small but non-zero entropy floor $E \approx 0.05$ nats, $\sim 35\times$ below the natural-language entropy of 1.69 and $\sim 2.6\times$ below it once the $14\times$ length calibration is applied. We interpret this near-vanishing floor of an irreducible entropy through the lens of algorithmic information theory: once the goal type and elaboration context are fixed, most of the next token carries little information.

Reading the residual. The histogram of Figure 4 is sharply concentrated: 57.6% of positions sit below 10^{-4} nats and 88% below 10^{-2} nats; the remaining mass decays monotonically across every subsequent bin, so only 0.94% of positions carry more than 1 nat and 0.33% more than 2 nats, mostly forced with a thin, monotonically-decaying right tail. The mean of 0.04 nats is set by that sparse tail, not by a broadly uncertain bulk: a rough bin-weighted decomposition places the bottom 88% of positions (below 10^{-2} nats) at under 1% of total entropy mass, the top 0.94% (above 1 nat) at roughly 44%, and the very top 0.33% (above 2 nats) alone at roughly a quarter. The tail is where the proof's content lives: which lemma to invoke, which case to split, which existential to inhabit; the deterministic plurality follows from the elaboration and the goal type. The contrast with the two NL corpora is stark. WikiText-103 (mean 2.79 nats, 64.5% above 2 nats) and arXiv abstracts (mean 1.73 nats, 38.0% above 2 nats) place their mass exactly where the Lean histogram has its sparse tail, and Lean places its mass where they have almost none.

Reading the boundary. The per-cell-best methodology behind the fit was forced by the data: at 16 epochs, the two largest cells overshot their test minimum before reaching the token budget, overfitting the data. 100M model with 16 epoch overshot by roughly 10% in log-eval space (minimum 0.157 at step 1275, the next saved checkpoint at step 1400 already at 0.172); Reading the per-cell minimum recovers the true asymptote, but the 16-epoch row would have collapsed the scaling law fit (Muennighoff et al., 2023). We

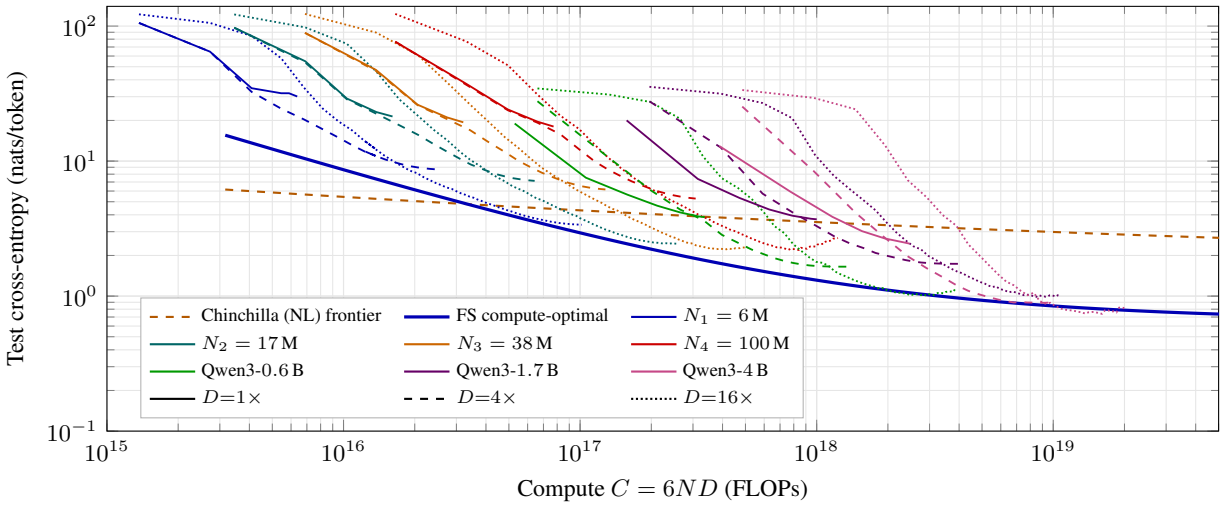

Figure 3. Test dataset cross-entropy against compute $C = 6ND$ for the twelve sweep cells, log–log. Our test cross-entropy is *length calibrated* — multiplied by 14, the proof-term verbosity of Table 2 — so the loss axis is comparable per natural-language token; the Chinchilla frontier is already on that basis. Each thin curve traces one cell's eval CE through training; colour denotes parameter count, dash pattern denotes token budget. Solid line: compute-optimal frontier $L^*(C)$ from the joint 21-cell fit of Table 3 (From-Scratch (FS) surface, $m_w = 1$), at which we have insufficient data to reach. Dashed line: the corresponding natural-language frontier of Hoffmann et al. (2022). The slope comparison is the point — and it is exactly what survives calibration, since rescaling every loss by a constant leaves the exponents unchanged.

	E	A	α	B	β	$R^2_{\log}$
This work, joint 21-cell fit (12 FS + 9 T)	**0.046**	3.88×10^7	**1.25**	6.71×10^7	**0.96**	0.988
NL ref. (Hoffmann et al., 2022)	1.69	406	0.34	411	0.28	—

Warm-start multipliers m_w on the data axis:

m_{FS}	1 (fixed)	from-scratch reference
$m_{0.6\mathrm{B}}$	**6.41**	Qwen3-0.6 B warm-start
$m_{1.7\mathrm{B}}$	**6.41**	Qwen3-1.7 B warm-start (collapses onto $m_{0.6\mathrm{B}}$)
$m_{4\mathrm{B}}$	**14.50**	Qwen3-4 B warm-start

Table 3. Unified joint fit $L(N, D, w) = E + A\,N^{-\alpha} + B\,(m_w\,D)^{-\beta}$ on all twenty-one cells (twelve from-scratch and nine warm-started; the warm-start enters as the data-axis multiplier m_w, fixed at 1 for FS), against the natural-language reference of Hoffmann et al. (2022). The exponents α, β are calibration invariant: the $\times 14$ loss rescaling that produced the scaling law chart in Figure 3 changes E, A and B but leaves α and β fixed. The upper row reads the joint 21-cell fit (the one equation used in both §3.1 and §3.2). The exponents come out roughly 3–4× steeper than natural language; the warm-start multipliers m_w are reported beneath the main row and read in §3.2.

therefore report each cell at its per-cell test-loss minimum rather than at the end of training.

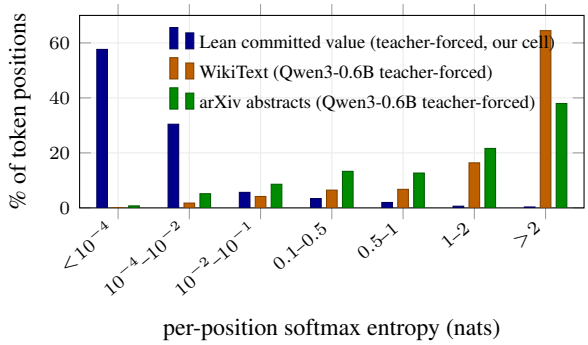

Figure 4. Most of a Formal Proof is Forced. We measure per-position softmax entropy from a teacher-forced forward pass across three corpora; the Lean curve is read from our from-scratch 100 M cell, the WikiText-103 and arXiv curves from Qwen3-0.6 B. **Lean Expr (blue):** 3.09 M token positions over 4,000 held-out committed-value rows; Mean 0.04 nats, median $< 10^{-4}$ nats, a 57.6% deterministic spike below 10^{-4} nats and a thin 0.94% tail above 1 nat. **WikiText-103 (orange):** 200k positions, mean 2.79 nats, median 2.64 nats, 64.5% of positions above 2 nats, 0.05% deterministic. **arXiv abstracts (green):** 200k positions, mean 1.73 nats, median 1.43 nats, 38.0% above 2 nats.

3.2. Transfer learning collapses to effective data

Claim. Beyond the grammar-capacity floor (the smallest N at which the parameter term $A\,N^{-\alpha}$ becomes a negligible share of L; see Table 4, approximately $N \geq 30$ M) the parameter term $A\,N^{-\alpha}$ is numerically negligible, so the law collapses arithmetically to

$$L(D, w) \;\approx\; E \;+\; B \cdot \big(m_w \cdot D\big)^{-\beta}, \qquad (6)$$

with a warm-start model multiplier m_w ($m_{\mathrm{FS}} = 1$ recovers from-scratch). A model halving its parameter count at a fixed effective data should land on the same curve. w indexes the model class from-scratch (FS), Qwen3-0.6 B, 1.7 B, and 4 B in our sweep, and m_w is its data-axis multiplier, fixed at 1 for FS and fitted otherwise.

Reading the formula. The joint fit (5) on all experiments is presented in Table 3 of Section 3.1; this section reads the warm-start multipliers it returns. From-scratch is the reference ($m_{\mathrm{FS}} = 1$); the small-Qwen3 bodies collapse onto a single $m_{0.6\mathrm{B}} = m_{1.7\mathrm{B}} = 6.41$, and the 4 B body sits at $m_{4\mathrm{B}} = 14.5$. The "above the floor" simplification of (6) is justified arithmetically from Table 4.

Test. A second nine-cell sweep reads the same surface with the model body initialised from a Qwen3 (Yang et al., 2025) pretrained checkpoint rather than from scratch: three model bodies (Qwen3-$\{0.6, 1.7, 4\}$ B) by the same three token budgets as before, otherwise identical recipe; because our 8,192-token vocabulary is not Qwen3's, the embedding table is rebuilt by the first-piece rule of Section E. Together with the twelve from-scratch cells the joint sweep is twenty-one cells.

N	$A\,N^{-\alpha}$ (nats)	typical fraction of L
6 M (FS)	0.142	9–55% (matters)
17 M (FS)	0.036	2–19%
38 M (FS)	**0.013**	**< 1–8%**
100 M (FS)	**0.004**	**< 1–3%**
440 M (Qwen3-0.6 B)	6×10^{-4}	$\sim$0.5%
1.4 B (Qwen3-1.7 B)	1×10^{-4}	$\sim$0.1%
3.6 B (Qwen3-4 B)	4×10^{-5}	$\sim$0.05%

Table 4. Numerical contribution of the parameter term $A\,N^{-\alpha}$ in the joint fit (Equation (5), coefficients in Table 3). Above the $\sim$ 30 M grammar-capacity floor (boldface rows) the term is arithmetically negligible, justifying the simplified $L \approx E + B(m_w D)^{-\beta}$ form of (6) without refitting on a subset.

The design was motivated by Hernandez et al. (2021), who absorb pretraining as an additive effective-token discount D_F on the data axis of the same Chinchilla form; the cells we measured do not fit that additive form; Section D records the eight forms we fit and the parsimony rule that carries this one.

The gap that carries the claim. The two smaller Qwen3 bodies deliver the same effective-data multiplier; the 4 B body delivers roughly 2.3× as much ($m_{4\mathrm{B}} = 14.5$ vs $m_{0.6\mathrm{B}} = m_{1.7\mathrm{B}} = 6.41$). One reading is that the 4 B body has internalised more formal-language structure during Qwen3 pretraining than the 1.7 B body, and that the gap is what the data-axis multiplier reads.

3.3. Discrete proof search on a sparse, heavy-tailed surface

Claim. The kernel-grounded surface, with its natural branching point `<step_end><accept>` (perhaps along with its sparse, heavy-tailed entropy), makes Monte-Carlo search-based algorithms like AlphaZero / AlphaProof-style search (Silver et al., 2018; Trinh et al., 2024) cheap in a specific sense. Instead of one model call per token, the model itself proposes a discrete schedule of branch points (`<step>` boundaries it emits in the chain of thought) the kernel prunes each step body locally, and rollout between branches is free.

Why this surface admits a different search shape. Beam search over prose has to budget at every token, because every prose token carries appreciable entropy and pruning the wrong one closes off a viable trajectory. Search over `Lean.Expr` does not face that constraint. At a forced position, the $\arg\max$ of the model's next-token discrete distribution is its near-certain choice and the remaining

probability mass is negligible. A search budget can, in principle, concentrate branching on the tokens with heavier probability mass alone.

Greedy decoding as a floor. Even without any search, the largest from-scratch cell (100M, 16 epochs) accepts $\mathbf{11/199}$ ($\mathbf{5.5}\%$) stratified held-out prompts under greedy commit-KAR[5] — a non-zero seed reachable by supervised fine-tuning alone, at the token level.

Depth-first search with the kernel in the loop. The decoder we actually run does not threshold on entropy. The model proposes branch points by emitting <step_begin> ...<step_end> spans during decoding; the kernel verifies each step body. On accept, decoding continues from a cached <step_end><accept>. On reject, the KV cache is rewound to the position immediately before <step_begin> and the step is resampled. Under 65,536 token-forwards, at most four commit attempts per problem, and sampling temperature 1, the warm-started Qwen3-0.6 B cell accepts $\mathbf{17/48}$ held-out problems — a $\mathbf{35.4}\%$ solve rate, 10 of which lie among the 26 Mathlib problems. Without reinforcement learning, it is direct evidence that the searching capability provided by the free rollout between branches converts into kernel-accepted proofs.

4. Related Work

Neural theorem proving over Lean. The dominant line generates tactic strings that drive a proof state: the GPT-f lineage (Polu & Sutskever, 2020; Han et al., 2022), LeanDojo and its retrieval prover (Yang et al., 2023), Llemma's proof-corpus pre-training (Azerbayev et al., 2024), DeepSeek-Prover's expert iteration (Xin et al., 2024), HyperTree proof search (Lample et al., 2022). These systems optimise pass-at-k on benchmarks, and the kernel is consulted at the end as a tactic's success flag, not as the direct metric on the produced object. We change the prediction target: the model emits the kernel's own representation, and the kernel is the metric. Closer in target are autoformalisation systems that emit a type-level statement directly (Wu et al., 2022; Jiang et al., 2022); what we add is the kernel-acceptance commitment as the primary metric and a measurement systematic enough to read a representation effect off it.

Scaling laws. We borrow the $L(F) = E + AF^{-\alpha}$ form from Hoffmann et al. (2022), and fit it per representation rather than collapsing across the comparison, so the exponent itself becomes a representation signal. The transfer section absorbs pretraining as a data-axis multiplier rather than the additive effective-token discount of Hernandez et al. (2021); the two forms are empirically distinguishable above the grammar-capacity floor (the data prefers a constant m_w to a constant D_F; see Section D).

Step-wise verdicts. Process-reward modelling trains a separate judge to score intermediate steps (Lightman et al., 2024; Uesato et al., 2022). Our design replaces the learned judge with a kernel call and keeps the verdict out of the loss, so the model learns the empirical verdict as context instead of a target. Our design is the kernel analogue of executing-as-you-generate (Key et al., 2022).

Information theory of language. How much a language fixes its next symbol is Shannon's question (Shannon, 1951); he put the entropy of printed English near a bit per character and its redundancy between one half and three quarters. Hindle et al. (2012) placed programming languages above prose on that axis, code is regular enough that plain n-gram models predict it well, and the de Bruijn factor (Wiedijk, 2000) measures the same redundancy as a coding cost, the size of a formalised proof against its informal source. Open code-LM evaluations such as Rozière et al. (2024) sit roughly an order of magnitude below natural-language references on the same axis. What we add is to carry Shannon's measurement onto formal mathematics and draw out what it implies for a prover.

5. Conclusion

Position. Natural-language proofs share a fixed axiomatic base, and their prose form is essentially a path through that base ornamented for human readers; removing the prose layer leaves the path itself. The Church Machine is therefore an ablation on whether natural language is a prerequisite for logical inference or one carrier among several — and the measurements report what falls out when the carrier is dropped.

This paper takes a position and reports one piece of evidence for it. The position: a sound way to grade a model that writes mathematics is to hand its output to the kernel of a proof assistant, and a productive surface to generate on is the kernel's own data structure rather than its human-facing source syntax. We demonstrate our paradigm with a unified 21-cell joint scaling-law fit $L = E + AN^{-\alpha} + B(m_w D)^{-\beta}$ (Equation (5), Table 3) that reads a small floor $E \approx 0.05$ nats against ~ 1.69 for natural language. We demonstrate that our training bottleneck is data size, while the model's benefit diminishes beyond 30M (Table 4). Under supervised fine-tuning alone the kernel already accepts proofs with 5.5% of 199 held-out commits on a from-scratch checkpoint under greedy decoding, 35.4% of 48 held-out problems on a warm-started cell once depth-first search puts the kernel

[5] Commit-KAR applies Equation (1) to the single witness term the model emits after <commit>, distinct from the per-step kernel verdicts inside the chain of thought. Here under greedy, $k = 1$.

in the loop. What this paper does *not* claim is a competitive prover. We have not run reinforcement learning, we have not exhausted the discrete-search algorithm space. The fundamental empirical finding is that the kernel accepts proofs even under this minimal regime.

Future work. The next steps are concrete: we can distil the effective data from overparameterised, fine-tuned, pre-trained models (e.g., 8 B or 14 B) into a smaller model, as our scaling law indicates that parameter count is not the bottleneck. Or, we can extend the discrete search in Section 3.3 beyond the single configuration measured here and close the loop with kernel-as-value-function reinforcement learning. Grammar-constrained decoding that masks invalid `Lean.Expr` continuations during training and inference is a further strengthening suggested in the review, and one we leave to future work. Because the exponents are calibration invariant (Section C), such a recoding is expected to shift the loss scale but not the slopes, thereby making the redesign a direct test of this paper's central claim.

Limitations. There is room for statistical improvement. First, the scaling fit reads < 4B non-embedding parameters. Second, the extrapolation past the largest cell is a fit, not a measurement. Third, the commit-acceptance numbers are read on 199 held-out prompts at one greedy attempt and 48 problems under depth-first search; both are modest held-out sets. The corpus draws from a single library ecosystem (Mathlib plus its standard-library neighbourhood and two competition-problem extracts); intra-corpus circularity, the model citing a lemma derived from the goal, is a hazard we mark but do not yet quantify how often an accepted proof leans on a library lemma definitionally close to its goal, so we cannot yet separate genuine search from near-neighbour recall. NL-trained provers like DeepSeek-Prover currently benefit from a much larger D in natural language. Beyond corpus scope, three deficiencies are worth stating plainly. **(i) No baselines.** We report no head-to-head comparison against tactic-string provers, Lean source-syntax generation, classical ATP, or MCTS search; the contribution is a representation-and-scaling measurement rather than a leaderboard result, and such comparisons on established benchmarks such as miniF2F are future work. **(ii) Underpowered solve rates.** The 5.5%-of-199 and 35.4%-of-48 figures are a proof-of-concept seed, too small to support paradigm-level claims about prover strength; the load-bearing evidence of this paper is the 21-cell scaling fit, not the solve rate. **(iii) Teacher-forced entropy.** The histogram and the "forced" fraction are measured under teacher forcing on gold prefixes, which upper-bounds the forcedness a decoder can exploit; whether the model identifies forced positions on its own rollouts is not directly measured, so the cheap-search claim is an amortised-cost argument about the surface, not a measured property of free generation.

6. Acknowledgement

This work was supported by the Singapore Global AIVP Programme (Award No. AIVP-2024-002) and NRF RSS Scheme (NRF-RSS2022-009). We would like to acknowledge that the computational work involved in this research is partially supported by NUS IT's Research Computing group under grant number NUSREC-HPC-00001 and NTU's CCDSAAI cluster.

References

Azerbayev, Z., Schoelkopf, H., Paster, K., et al. Llemma: An open language model for mathematics. In *ICLR*, 2024.

Coquand, T. and Huet, G. P. The calculus of constructions. *Information and Computation*, 1988.

Curry, H. B. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, 1934.

de Moura, L. and Ullrich, S. The Lean 4 Theorem Prover and Programming Language. In *CADE-28*, 2021.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. In *ICLR*, 2022.

Hernandez, D., Kaplan, J., Henighan, T., and McCandlish, S. Scaling laws for transfer. arXiv:2102.01293, 2021.

Hindle, A., Barr, E. T., Su, Z., Gabel, M., and Devanbu, P. On the naturalness of software. In *Proc. International Conference on Software Engineering (ICSE)*, 2012.

Hoffmann, J., Borgeaud, S., Mensch, A., et al. Training compute-optimal large language models. In *NeurIPS*, 2022.

Howard, W. A. The formulae-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. 1980.

Jiang, A., Li, W., Tworkowski, S., Czechowski, K., Odrzygóźdź, T., Miłoś, P., Wu, Y., and Jamnik, M. Thor: Wielding hammers to integrate language models and automated theorem provers. In *NeurIPS*, 2022.

Kaplan, J., McCandlish, S., Henighan, T., et al. Scaling laws for neural language models. arXiv:2001.08361, 2020.

Key, D., Li, W.-D., and Ellis, K. I Speak, You Verify: Toward trustworthy neural program synthesis. In *ICLR Workshop*, 2022.

Lample, G., Lacroix, T., Lachaux, M.-A., et al. HyperTree proof search for neural theorem proving. In *NeurIPS*, 2022.

Lightman, H., Kosaraju, V., Burda, Y., et al. Let's verify step by step. In *ICLR*, 2024.

Lin, Y., Tang, S., Lyu, B., et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. arXiv:2508.03613, 2025.

Liu, J., Zhou, Z., Zhu, Z., et al. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. arXiv:2601.14027, 2026.

Martin-Löf, P. An intuitionistic theory of types. In *Twenty-five years of constructive type theory*. 1998.

Muennighoff, N., Rush, A. M., Barak, B., et al. Scaling data-constrained language models. In *NeurIPS*, 2023.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. arXiv:2009.03393, 2020.

Ren, Z. Z., Shao, Z., Song, J., et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. arXiv:2504.21801, 2025.

Rozière, B., Gehring, J., Gloeckle, F., et al. Code llama: Open foundation models for code. arXiv:2308.12950, 2024.

Shannon, C. E. Prediction and entropy of printed English. *Bell System Technical Journal*, 1951.

Silver, D., Hubert, T., Schrittwieser, J., et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 2018.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 2024.

Uesato, J., Kushman, N., Kumar, R., et al. Solving math word problems with process- and outcome-based feedback. arXiv:2211.14275, 2022.

Wiedijk, F. The De Bruijn factor. https://www.cs.ru.nl/~freek/factor/, 2000.

Wu, Y., Jiang, A. Q., Li, W., et al. Autoformalization with large language models. In *NeurIPS*, 2022.

Xin, H., Guo, D., Shao, Z., et al. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale synthetic data. arXiv:2405.14333, 2024.

Yang, A., Li, A., Yang, B., and Others. Qwen3 technical report. arXiv:2505.09388, 2025.

Yang, K., Swope, A., Gu, A., et al. Leandojo: Theorem proving with retrieval-augmented language models. In *NeurIPS*, 2023.

Ying, H., Wu, Z., Geng, Y., Wang, J., Lin, D., and Chen, K. Lean Workbook: A large-scale Lean problem set formalized from natural language math problems. In *Advances in Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2024.

Zhao, Y., Qu, Y., Staniszewski, K., Tworkowski, S., Liu, W., Miłoś, P., Wu, Y., and Minervini, P. Analysing the impact of sequence composition on language model pretraining. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7897–7912. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.427. URL http://dx.doi.org/10.18653/v1/2024.acl-long.427.

A. Difference of Reasoning in Lambda Calculus

The work closest to ours, e.g. the DeepSeek-Prover family (Xin et al., 2024; Ren et al., 2025) and Goedel-Prover (Lin et al., 2025), differs from the Church Machine not in ambition but in *medium*.

The concrete differences set the two apart:

1. *Generation surface.* Church Machine emits `Lean.Expr` AST tokens directly — the kernel's own constructors. DeepSeek-Prover emits Lean surface-syntax strings, parsed and elaborated only after the fact.

2. *Supervision target.* Loss is taken over the elaborated `Lean.Expr` together with per-step kernel verdicts (accept/parse_err/type_err/neq) the kernel itself provided; the verdict distribution is in the SFT loss itself. DeepSeek-Prover backpropagates kernel signal only through an RL outer loop on tactic-string rollouts, not into the per-token cross-entropy.

3. *Search granularity.* Depth-first kernel-grounded search (Section 3.3) costs one model call per *branch point*, with chain-rollout free between them; DeepSeek-Prover's RMaxTS variant samples whole proofs per call, with no equivalent of free chain-rollout between branch points.

Our theorem-proving method is not confined to natural language, which may enable us to construct proofs that are difficult to articulate in natural language or to design search methods with complexity that is impossible to implement under natural-language entropy.

B. Shannon's Measurements & the de Bruijn Factor

`Lean.Expr` is essentially a typed lambda calculus, which is Turing-complete and has three grammatical cases: variable, application and abstraction. Such minimality is capable of carrying every high-level mathematical idea and proof. Since it is such a low-level programming language, representing high-level programs in it results in massive combinatorial expansion. In this sense, the lambda calculus is a verbose language. The analysis below is therefore information theoretic: the per-token Shannon entropy is low, not because the grammar is syntactically redundant, but because each token carries little Kolmogorov complexity once the elaboration context is in hand. A high-Kolmogorov decision point is rare; a long stretch of forced unfolding is the common case.

Shannon's measurement, and ours. Shannon (1951) asked how much of printed English is fixed by what precedes it, and answered with a number: against the entropy $H_{\max} = \log_2 27 \approx 4.76$ bits an uniform alphabet (with spacing) would carry, English runs at one to one-and-a-half bits per character — a *redundancy* $R = 1 - H/H_{\max}$ of one half to three quarters, depending on how much context the predictor is given. Redundancy is the fraction of the channel spent on structure the receiver could have reconstructed for itself, and it is the natural axis on which to place a formal language. Plain n-gram models predict code better than prose (Hindle et al., 2012) because a compiler, unlike a reader, rejects most strings outright. A kernel-grounded proof is the far end of that ladder. Measured with the model of Section 3.3 as an entropy upper-bound predictor (a single warm-started checkpoint on 48 held-out problems, 2.19 M sampled token positions), the conditional entropy of `Lean.Expr` comes in at ~ 0.11 nats, about 0.16 bits, against the $\log_2 8192 = 13$ bits its vocabulary could carry — an upper bound on a redundancy ratio of ~ 0.99. Formal proof on this surface sits well above English on Shannon's ladder.

The de Bruijn factor. Redundancy has a second face, visible without a model at all. A proof carries some fixed quantity of mathematical information; a serialisation spends some number of tokens conveying it; the ratio is a coding efficiency. The formal-methods literature has long measured it — the *de Bruijn factor*, the size of a formalised proof against its informal original, which Wiedijk (2000) found to sit near four across systems and texts. Our serialisation is more verbose still: Table 2 puts a proof term at roughly $14\times$ the tokens of its natural-language rendering. The gap from Wiedijk's four is not slack we left in — it is elaboration. The kernel demands a fully explicit term, so every implicit argument, every type-class instance, every universe level a human leaves to inference is materialised in the tree the model must emit. The $14\times$ is the de Bruijn factor of a *maximally elaborated* surface — and the near-zero entropy floor of Section 3.1 is that same number read as a loss: spread a fixed quantity of information across $14\times$ the tokens, and the per-token figure falls in proportion.

One property, three readings. The rest of this section investigates the consequences of a near-deterministic source. Section 3.1 confirms the shape directly — a per-position entropy histogram with most of its mass concentrated near zero and a thin heavy tail of genuine decisions, a sparse two-regime shape rather than the uniform case — and shows that the scaling exponents which follow are steep and, unlike the loss scale, invariant to the verbosity just described. Section 3.2 shows that, the grammar being so nearly forced, model size becomes irrelevant once it is large enough to hold that grammar, and loss is then determined by data alone. Section 3.3 turns the concentration into solved problems: a search that ignores forced positions and concentrates its branching budget on the sparse decision points converts the entropy shape into accepted proofs. A high-redundancy language is a liability for a compressor but an exploitable opportunity for a prover; the three measurements are that single observation put in turn to the model, the optimiser, and the search.

C. Information-Theoretic Framing

This appendix states the framework of Section B precisely.

Proposition 1 (Information is conserved; per-token loss is not). Let a proof carry mathematical information I — the entropy of the goal-conditioned distribution over proofs that inhabit it. A serialisation encodes that proof in T tokens, and the cross-entropy any model can achieve, summed over those tokens, is bounded below by I. Per-token loss is therefore at least I/T: it falls as the serialisation lengthens. Elaboration is a length-multiplying and information-preserving recoding. It materialises implicit structure the kernel re-derives, so a de Bruijn factor of c divides the per-token floor by c. The near-zero E measured on our $14\times$ surface and the $14\times$ verbosity itself are therefore not two findings but one.

Proposition 2 (The exponents are calibration-invariant; the floor is not). Write the fit as $L(N, D) = E + AN^{-\alpha} + BD^{-\beta}$. A recoding that rescales every measured loss by a constant, $L \mapsto cL$, sends $(E, A, B) \mapsto (cE, cA, cB)$ and leaves α and β untouched: on log–log axes a constant factor is a vertical shift, and a shift does not change a slope. The steep exponents therefore survive any tokenisation calibration and the near-zero floor does not — the slopes are the representation-invariant content of the from-scratch fit, the

floor its calibration-dependent part.

Proposition 3 (The entropy is bursty, and search costs accordingly). Model the token stream as a mixture of two regimes: *forced* positions, at which the goal type and context leave one continuation (entropy ≈ 0), and *choice* positions, at which they do not. The entropy rate is then $h = p_{\text{choice}} \cdot \bar{H}_{\text{choice}}$, and the histogram of Section 3.3 estimates p_{choice} small — a tail of 0.94% above one nat. A search that branches only at choice positions and copies the forced remainder has a cost set by their *count*, not by proof length: this is the quantitative content of one model call per branch point.

Proposition 4 (Cross-entropy grades strings; the kernel grades terms). Cross-entropy scores the exact token sequence; the acceptance metric of Equation (1) scores the kernel object up to definitional equality. For a goal in Prop, proof irrelevance collapses every inhabitant into a single `isDefEq` class, so many distinct token strings are kernel-equivalent and a model's token-level cross-entropy *over-counts* the uncertainty the task actually carries. The gap between a non-trivial test cross-entropy and a non-zero commit-acceptance rate is, in part, this quotient — the asymmetry between Prop and Type goals already drawn in Section 1.

D. Forms Considered for the Transfer Law

The transfer law of Equation (6) (Hernandez et al., 2021) (the multiplicative re-statement of the additive form in Hernandez et al. (2021); the per-cell fit comparison below selects it) is one of a family we fit to the joint from-scratch and transfer data. We record the family here so the carried form is on the record as a selection.

The joint dataset is twenty-one cells — the twelve from-scratch cells of Section 3.1 and the nine warm-started transfer cells of Section 3.2 — each read at its per-cell test minimum, with the D=16 cells down-weighted by one half (the data-constrained corner of Section 3.1). Every form is fit by weighted least squares in log space, and selected by parsimony: weighted $R^2_{\log}$ read against the free-parameter count, not on its own.

H3, carried. The warm-start enters as a gain on the data axis — one shared exponent β, a per-warm-start-class multiplier m_w on D. It is the best fit at its parameter count, and the multiplier reads cleanly: a warm-start worth m_w epochs of from-scratch data. Past the capacity floor it is the law reported in the body.

H4 scores higher and is still not carried. H4 reaches $R^2_{\log} = 0.985$ against H3's 0.984 — but at nine free parameters against eight, the ninth a separate data exponent for the transfer surface. Twenty-one cells do not resolve a second exponent, and no mechanism asks for one: the surfaces differ in how much effective data the warm-start supplies, not in how loss falls with data. A gain of 0.001 does not buy the parameter. H4 would become the right form if a larger sweep showed the two surfaces' data exponents genuinely apart.

H5 is a ceiling, not a law. H5 fits each surface its own A, α, β — eight parameters, no shared structure. It models nothing about transfer; it is two fits printed together. Its use is as an upper bound on what any unified form could reach, and H3 meets it (0.984 against 0.980).

The additive forms, H0–H2. H0 gives the transfer surface its own exponent but no warm-start term, and fits it worst of the unified family ($R^2_{\text{T}} = 0.875$). H1 adds an additive offset — a fixed effective-token dowry D_F per class — and does not improve on H0. H2 constrains that offset to $k\,N^p$ and fits worst of all. Additive and multiplicative are the same object under $D_F = (m_w - 1)\,D$; the data preferring a constant m_w to a constant D_F is the content of the multiplicative finding.

Dropping the size term, H3′/H4′. Past the ~ 30 M capacity floor the $A\,N^{-\alpha}$ term is negligible, so removing it should cost nothing — and on the transfer surface it does not (H3′ holds $R^2_{\text{T}} = 0.974$). Across all twenty-one cells R^2_{FS} falls to 0.938: the two smallest from-scratch cells, at 6 and 17 M, sit below the floor and still need the term. H3′ is thus not rejected but regime-specific — restricted to $N \geq 30$ M it is the arithmetic simplification of Equation (6), justified by Table 4. Within that restriction the same parsimony rule prefers a two-regime multiplier — m shared across the 0.6 and 1.7 B warm-starts, distinct for 4 B — over a free per-class one: equal R^2, one parameter fewer.

E. Warm-Start Embedding Transfer

The transfer cells of Section 3.2 initialise the Transformer body from a Qwen3 checkpoint, but Qwen3's tokeniser and ours share no vocabulary, so the embedding and output matrices cannot be copied across. We rebuild them for our 8,192-token vocabulary by a *first-piece* rule. A reserved special with a clean Qwen3 analogue — `<eos>`, `<commit>`, the step markers — takes that Qwen3 token's exact row. Every other token is re-encoded as a string under Qwen3's own tokeniser, and the row of the *first* resulting sub-piece is copied: the leading byte-pair piece is a semantically meaningful prefix the pretrained body has already been trained against, so the embedding starts as a usable prior rather than as noise. The few tokens whose surface form decodes empty fall back to $\mathcal{N}(0, 0.02^2)$, Qwen3's own initialisation scale.

	Form	k	$R^2_{\log}$	R^2_{FS}	R^2_{T}
H0	β per surface; no warm-start	6	0.972	0.961	0.875
H1	additive D_F per class	8	0.972	0.953	0.900
H2	additive $D_F = k\,N^p$	7	0.966	0.952	0.852
H3	**multiplicative $m_w D$ per class**	**8**	**0.984**	0.965	0.968
H4	additive D_F; β per surface	9	0.985	0.980	0.935
H5	surfaces fully decoupled	8	0.980	0.981	0.891
H3′	H3, no $A\,N^{-\alpha}$	6	0.975	0.938	0.974
H4′	H4, no $A\,N^{-\alpha}$	7	0.968	0.936	0.920

Table 5. Transfer-law forms fit to the joint twenty-one-cell dataset. All share the base $L = E + A\,N^{-\alpha} + B(\cdot)^{-\beta}$; the *Form* column gives the warm-start treatment of the bracket and any per-surface split. k counts free parameters; R^2_{FS} and R^2_{T} split the weighted fit over the from-scratch and transfer cells. **H3** is carried into Equation (6).

The rule covers essentially the whole vocabulary, and it is the only non-trivial adaptation the warm-start needs — the decoder blocks and the final norm copy across unchanged.

F. Compute Hardware and Numerical Identity

All cells were trained in BF16 on a heterogeneous pool of NVIDIA GPUs — a shared multi-tenant cluster with A100 (40 and 80 GB), A6000 (48 GB), and H100 (80 GB) partitions, where the device allocated for a given cell was determined by memory fit and queue availability rather than uniformly assigned. To make the underlying mathematical operation identical *in distribution* across devices, four things are held fixed per cell: the global batch size in tokens (recovered via samples-per-step $\times$ gradient-accumulation when per-device batch shrinks under VRAM), the BF16 forward and backward with FP32 master weights and AdamW state, the seeded shuffle of the packed corpus, and a FlexAttention kernel with a block-diagonal mask over the intra-document pack; we did not attempt to suppress it, and we treat it as a contribution to the optimisation noise the cosine schedule already smooths over.

G. Use of Generative AI

Anthropic's Claude Code Opus was used as a writing aid: prose polishing, copy-editing, table layout, and citation cross-checks. All scientific claims, experimental design, and numerical results are the author's; the author has read every change and takes full responsibility for the final text. No LLM is listed as an author. This disclosure follows the ICML 2026 generative-AI policy, which encourages authors to explain notable uses of such tools.