LANGUAGE MODELLING AND SCALING LAWS FOR LEAN'S LAMBDA CALCULUS

Anonymous authors
Paper under double-blind review

ABSTRACT

Typed lambda calculi underlie many proof assistants; e.g., Lean's kernel checks terms that express proofs and programs, making proof steps explicit. How difficult is this language for an autoregressive model to learn and generate? We compare kernel terms, surface Lean, and informalised English descriptions of the same Mathlib theorems using our developed serialisation method, which follows the dependent type from left to right. The decoder then samples kernel expressions, which can be checked without elaboration. We probe the scaling laws, both from scratch and with a warm start using Qwen3, in each language. For Qwen3-4B warm starts under reference prefixes, the 1% of proof tokens with the highest loss carry 98.6% of term proof loss, against 10.7% for surface Lean and 14.3% for English. Mean term proof loss is 2.475 times surface Lean's at these warm starts but falls faster with model size and data from scratch, though the grid's single learning rate slightly inflates the term model-size exponent. If the fits continue with fresh data and model size taken to infinity, term proof loss crosses below surface Lean's after about 7 times the current pool of distinct proofs. The transferred Qwen3-0.6B achieves a term proof loss lower by a factor of 3.68 than the same network trained from scratch on the same number of tokens. Based on these results, we're in a position to use pretrained models and increase the dataset size for kernel term language.

1 INTRODUCTION

We ask what happens when a model learns a language with a minimal grammar, the lambda calculus. Lean's type system is based on lambda calculus, which can state and prove mathematical statements and describe programs (Carneiro, 2019; de Moura & Ullrich, 2021). Its precision and the ability to check proofs with a small kernel motivate our study: this language could help models carry out mathematical and programming work. A short Lean proof can stand for a long derivation. A mathematician may write `by simp` and let Lean find the intermediate steps. Lean then produces a kernel term: an explicit proof that its small trusted kernel can check against the theorem's statement. For a Qwen3-4B model transferred to terms (a warm start), the difficulty lies in a few choices per proof, such as which premise to name: under reference prefixes, half the proof loss sits in a few tokens (median 4.0) within kernel terms thousands of tokens long (median 3,034, including the statement).

We compare three languages for the same mathematical content: kernel terms, the surface Lean written by users, and informalised English descriptions generated from that source by a language model. English explains an argument; surface Lean can ask tactics to construct it; a kernel term records a proof that can be checked directly in its library environment. Lean expresses both statements and proofs with the twelve constructors of the `Lean.Expr` type, over the `mathlib` of names (Appendix B).

Earlier neural studies measured task accuracy on generated symbolic expressions and small lambda-calculus fragments (Lample & Charton, 2020; Flach et al., 2026; Sekiyama & Suenaga, 2018). Library-scale elaborated terms have served as auxiliary training data (Han et al., 2022), while a non-archival workshop abstract modelled and sampled pretty-printed Lean proof terms (Palermo et al., 2022). The closest prior work is the non-archival workshop paper by Yeh & Ong (2026), which fitted a preliminary scaling law to terms alone. Most neural provers instead generate tactics or complete Lean source proofs during search (Lample et al., 2022; Yang et al., 2023; Xin et al.,

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

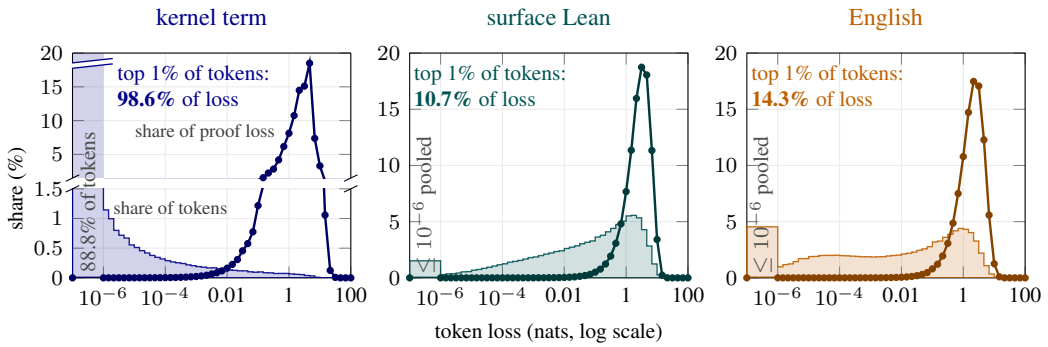

Figure 1: Most term proof tokens have low prediction loss, while a small fraction carries most of the loss. Bars show the share of tokens in each loss bin on a logarithmic axis; the line shows their share of proof loss. Shares are pooled over held-out proof tokens; within a proof, half the loss sits in a median of 4.0 tokens for terms, 2.0 for surface Lean and 13.0 for English. Scores use Qwen3-4B models transferred to each language (warm starts), teacher-forced on the reference proof.

Table 1: Contributions.

Conceptual	To our knowledge, the first same-theorem comparison of proof loss, its distribution and scaling across kernel terms, surface Lean and informalised English (Section 3.1).
Methodological	We developed a serialisation of whole Mathlib statements and proofs whose Lean 4 kernel terms fit the context. One left-to-right decoding pass turns samples into kernel expressions for checking without elaboration (Section 3.2).
Empirical	Under reference prefixes, most term tokens have little loss, while a few choices per proof, such as which premise to name, carry half (Fig. 1).
	Mean term proof loss is 2.475 times surface Lean's at the Qwen3-4B warm starts. From scratch, it falls faster with model size and data than surface Lean's, though the grid's single learning rate slightly inflates the term model-size exponent. If the fits continue with enough fresh data and model size taken, term proof loss crosses below surface Lean's after about 7 times the current pool of distinct proofs (Table G.3).
	In the low-loss regime, proof loss predicts checked term generation: the lower bound on pass@k from reference-proof probability sits just below measured success under independent unit-temperature sampling (Table 3).
	The post-trained Qwen3-0.6B model transfers from text to the lambda calculus with a new vocabulary, ending with term proof loss lower by a factor of 3.68 than the same network trained from scratch on the same tokens (Section 4.3).

2024; Lin et al., 2025). Scaling laws describe how language-model loss changes with model size and data (Kaplan et al., 2020; Hoffmann et al., 2022), but these lines of work leave open how loss is distributed and scales across terms, surface Lean and English for the same library theorems.

We train models on the same Mathlib theorems in these languages, measure prediction loss and check generation. We take the position that directly modelling kernel terms is a promising route to scalable verified proof generation. Adapting pretrained models and enlarging the pool of distinct proofs are concrete ways to pursue it, with kernel-checked success per unit of compute as the final test.Table 1 summarises our contribution.

2 Related work

Symbolic mathematics and lambda-calculus. Transformers solve generated problems in symbolic mathematics and school mathematics (Lample & Charton, 2020; Saxton et al., 2019). Lambda-calculus studies train transformers on small generated terms for untyped reduction (Flach et al., 2026) or simply typed type inference (Miranda et al., 2023). Sekiyama & Suenaga (2018) guide

search for propositional proofs represented as lambda terms, while Kusumoto et al. (2018) use reinforcement learning for propositional proof-tree search. These evaluations focus on task accuracy and proof-search success within restricted fragments.

Kernel terms at library scale. PACT co-trains a tactic prover with tasks on pretty-printed and fully elaborated Lean 3 proof terms, including proof-term prediction and elaboration (Han et al., 2022). Its extraction truncates deep subtrees, and its evaluation provides no analysis of term loss, scaling study or kernel-checked whole-term generation. Skip-tree training models HOL Light statements and intermediate goals as typed expressions, rather than proof objects (Rabe et al., 2021). GamePad encodes Coq kernel terms for tactic prediction and proof-state evaluation (Huang et al., 2019). Graph2Tac encodes Coq kernel graphs for tactic selection, omitting proof terms from its definition graphs (Blaauwbroek et al., 2024). Kogkalidis et al. (2024) encode elaborated Agda types for premise selection, and MLFMF exports elaborated Lean and Agda entries, including their bodies, for premise-recommendation benchmarks (Bauer et al., 2023). Roosterize uses elaborated Coq statement trees to generate lemma names (Nie et al., 2020). The closest language-modelling precedent is the non-archival AITP abstract of Palermo et al. (2022), which reports loss and kernel-filters samples from one model trained on short, pretty-printed Lean 3 proof terms.

Language models for formal mathematics. Neural provers generate tactics during proof search (Lample et al., 2022; Yang et al., 2023), Metamath proof steps (Polu & Sutskever, 2020), or complete Lean source proofs (Xin et al., 2024; Lin et al., 2025). Minilang simplifies Isabelle's surface proof language while retaining access to automation (Xu et al., 2026). Yeh & Ong (2026) studied next-token predictability and preliminary scaling in Lean kernel terms in a non-archival workshop paper. This paper compares complete serialised kernel terms with surface Lean and English for the same theorems, measures loss concentration, transfer and scaling, and generates checked whole terms. Our sampling leaves the model's distribution unchanged and checks completed outputs; grammar- or type-constrained decoding modifies the available continuations (Poesia et al., 2022; Mündler et al., 2025).

Transfer and scaling. Fast Vocabulary Transfer initialises new embeddings from the donor's token pieces (Gee et al., 2022); WECHSEL and FOCUS use semantic similarity for transfer between natural languages (Minixhofer et al., 2022; Dobler & de Melo, 2023). We apply vocabulary transfer to Lean constructors and names. Scaling laws describe how loss changes with model size and data (Kaplan et al., 2020; Henighan et al., 2020; Hoffmann et al., 2022). Luo et al. (2026) compare code with natural language, while Yang et al. (2026) compare programming languages. Our comparison holds the theorems fixed. We examine sensitivity to fitting choices (Besiroglu et al., 2024), training recipe (Porian et al., 2024), parameter counts (Pearce & Song, 2024) and vocabulary size (Tao et al., 2024).

Where prediction loss occurs. Previous work compares the predictability of code and natural language (Casalnuovo et al., 2019), measures loss on Coq scripts and kernel-level HOL Light proof traces (Hellendoorn et al., 2018), and studies high-entropy tokens in chain-of-thought reasoning (Wang et al., 2025). We measure reference-token surprisal: how much probability the model assigns to the observed token. This differs from the entropy of the full next-token distribution. Its sum gives both a proof's code length (Delétang et al., 2024) and the negative log-probability of reproducing that proof, which we connect to verified generation.

3 Methods

3.1 The same theorems in three languages

We extract declarations from Mathlib (The mathlib Community, 2020) revision `51e6992e`, using Lean `v4.33.0-rc2`. Each declaration has surface Lean source, a kernel term produced by elaboration, and English generated from the source and its enclosing scope by `Qwen/Qwen3.8-27B`. The generator produces an outline and a full write-up; we train on both and average their losses for each theorem. Thus, the English comparison concerns model-generated explanations of formal mathematics. The English prompt is reproduced in Appendix A.

Every model has a context of 65,536 tokens. We retain theorems whose term fits this context; their surface and English versions also fit without truncation. This cap removes 27,533 of 276,398

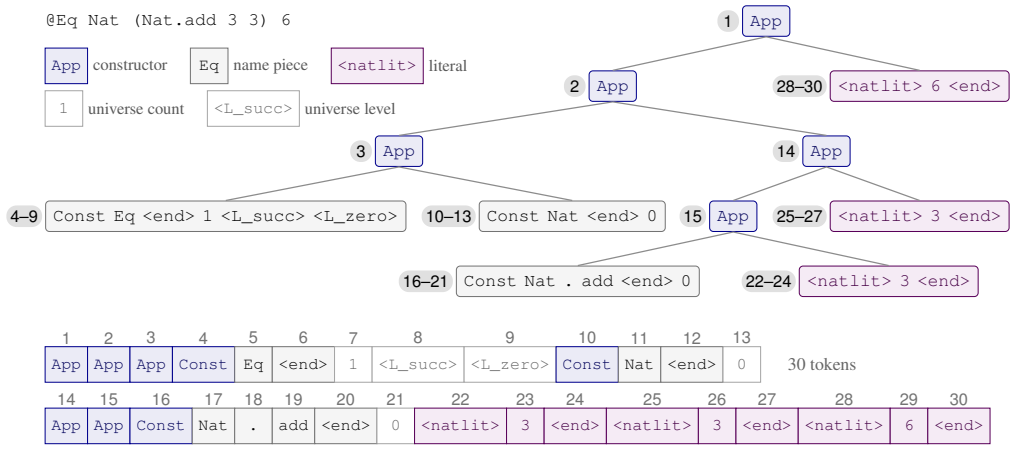

Figure 2: A Lean expression for `@Eq Nat (Nat.add 3 3) 6`, or $3 + 3 = 6$, and its pre-order sequence. Each node's badge gives the span of stream positions it occupies. The expression is built directly for illustration; elaborating Lean notation also introduces typeclass instances. The strip shows the model's 30 tokens, with split names, hexadecimal literals and explicit universe levels (Appendix C).

candidate theorems, and six other screens leave 175,252. The excluded proofs use automation more often: 71.6% of the matched excluded set (theorems removed by the cap alone), compared with 28.2% of the retained set.

A seeded hash of each Mathlib module name assigns the whole file to training or evaluation. The split gives 157,686 training theorems and 17,566 held-out theorems in 755 modules. Some proof text repeats across modules: 0.19% of held-out term proofs, 20.85% of surface proofs and 0.00% of English proofs match a training proof exactly. Short surface proofs such as `rfl` can be reused for many different statements. Appendix K reports refits after removing exact proof matches.

3.2 Turning proof trees into sequences

We serialise a kernel term in pre-order: each node comes before its children (Fig. 2). Binder types precede their bodies, and let bindings put the type and value before the body that uses them. This makes those dependencies available to causal attention. Bound variables are stored as de Bruijn indices, which record how far back their binders occur. Constructor arities and explicit boundaries let a decoder recover the tree from left to right without brackets. The serialisation preserves expression fields up to binder names, metadata and the unused non-dependence flag on let bindings (Appendix C). The vocabularies are 12,399 tokens for constructors and pieces of environment names, indices and literals; surface Lean and English use byte-level BPE vocabularies of 2,048 and 16,384 tokens.

For transfer, we use the released, post-trained Qwen3 checkpoints (Yang et al., 2025). Fast Vocabulary Transfer (Gee et al., 2022) initialises each new token embedding from the mean of the donor embeddings of its text pieces. We copy the transformer weights and continue training on our sequences. These warm starts are analysed separately from models trained from scratch.

3.3 Measuring prediction loss

The three languages express a theorem in very different numbers of tokens, so we compare their total prediction loss per theorem. For a model π, arithmetic coding approaches encode a sequence x as the ideal length of a code that uses the model to write that sequence down, which is the sequence's negative log-probability $\ell(x) = -\log \pi(x)$ within two extra bits[1] (Delétang et al., 2024). Averaging this length over theorems drawn from the data distribution p gives cross-entropy $\mathrm{CE}(p, \pi)$, with the average written $\mathbb{E}_{x\sim p}$. It equals the source's entropy $H(p)$, its minimum expected code length at $\pi = p$, plus the extra length from the model's prediction error, the Kullback–Leibler divergence $\mathrm{KL}(p \,\|\, \pi)$: $\mathrm{CE}(p, \pi) = \mathbb{E}_{x\sim p}[\ell(x)] = H(p) + \mathrm{KL}(p \,\|\, \pi)$ (Cover & Thomas, 2006). This code

[1]The logarithm is natural, so the unit is in nats, approximately 1.44 bits.

length in nats puts the three languages on one scale regardless of token boundaries, keeping the information-theoretic interpretation, whereas per-token loss depends on the tokenisation.

We write the token at position i as x_i and its preceding reference prefix as $x_{<i}$. Each theorem row has a statement block of n_x tokens and a proof block of m_x tokens:

$$x = (\underbrace{\texttt{<bos>}}_{x_0}, \underbrace{\texttt{<kind>}, \texttt{<comp>}, \ldots, \texttt{<sep_type>}}_{x_1, \ldots, x_{n_x}\text{: statement}}, \underbrace{\ldots, \texttt{<eos>}}_{x_{n_x+1}, \ldots, x_{n_x+m_x}\text{: proof}}). \tag{1}$$

The statement includes the declaration kind, computability and separator; the proof includes its closing <eos>. We train on every token after <bos>; padding is masked. We score each next token conditional on the preceding reference tokens, a procedure called teacher forcing (Williams & Zipser, 1989). The per-theorem proof loss $\ell_{\text{proof|stmt}}(x)$ sums the negative log-probabilities over the proof block, including its closing <eos>:

$$\ell_{\text{stmt}}(x) = -\sum_{i=1}^{n_x} \log \pi(x_i \mid x_{<i}) \tag{2}$$

$$\ell_{\text{proof|stmt}}(x) = -\sum_{i=n_x+1}^{n_x+m_x} \log \pi(x_i \mid x_{<i}). \tag{3}$$

The whole-theorem loss $\ell(x)$ is exactly the sum of the statement and proof losses. Taking expectations $\mathbb{E}_{x\sim p}$ gives the statement cross-entropy $\text{CE}_{\text{stmt}}(p, \pi)$ and the proof cross-entropy conditional on the statement $\text{CE}_{\text{proof|stmt}}(p, \pi)$, and preserves the exact split

$$\text{CE}(p, \pi) = \text{CE}_{\text{stmt}}(p, \pi) + \text{CE}_{\text{proof|stmt}}(p, \pi). \tag{4}$$

The mean proof loss over held-out theorems is the reported proof cross-entropy, in nats per theorem. Unless a region is named, reported losses and scaling fits use proof cross-entropy $\text{CE}_{\text{proof|stmt}}$. Appendix D gives the definitions and derivation.

The prompts express the same theorem but contain different information: surface statements retain names and attributes, while kernel types make implicit arguments and resolved context explicit. The comparison therefore measures these languages as used in the corpus.

3.4 Training and scaling fits

Models trained from scratch share a decoder-only transformer design and the same training recipe (Appendix E). Each language has a grid of model sizes and token budgets, with at most two exposures to each training token (Muennighoff et al., 2023). We fit the Chinchilla form (Hoffmann et al., 2022) separately to each grid:

$$L(N, D) = E + A\,N^{-\alpha} + B\,D^{-\beta}. \tag{5}$$

Here N counts total parameters, including embeddings, and D counts training tokens processed. The exponents α and β describe how the corresponding loss contributions decrease; E is a fitted asymptotic floor. Changing from nats per theorem to nats per token rescales each language's loss and leaves its exponents unchanged (Appendix L).

We fit in log space with exponent priors informed by published scaling laws and use a Laplace Bayesian approximation for posterior intervals. For the two Lean languages the priors are $\alpha \sim \mathcal{N}(0.466, 0.204)$ and $\beta \sim \mathcal{N}(1.232, 0.475)$, centred on the medians of published code fits. English uses the same widths around the fit of Hoffmann et al. (2022). Since the fitting objective, including the priors, is minimised at $E = 0$, we hold it when reporting exponents and their posterior intervals (Appendix K). Refitting the other parameters at each fixed floor gives the profile from which we report a one-sided 95% upper limit on E. We treat the fits as summaries of the observed grids and report their errors alongside the exponents.

4 Results

We first ask where the loss of a long proof occurs, then whether models can generate complete proofs, and finally how transfer and additional training improve prediction. The main results use the Qwen3-4B warm starts, one per language. From-scratch models at a common width provide the size-matched control across languages, with 168.8M, 160.9M and 171.9M parameters for terms, surface Lean and English. Each uses its language's largest training-token budget (Appendix E).

Table 2: Long sequences can have low loss per token and still have higher loss per proof. Proof lengths and losses on the same 17,566 held-out theorems; parentheses give medians. Nats per token pools proof tokens. From-scratch models share $d_{\text{model}} = 768$, about 169M parameters.

		nats per token		nats per proof	
language	proof tokens	Qwen3-4B	from scratch	Qwen3-4B	from scratch
kernel term	7,468 (2,056)	0.010 (0.011)	0.045 (0.056)	74.5 (27.5)	336.6 (117.7)
surface Lean	34 (20)	0.875 (0.750)	2.607 (2.382)	30.1 (15.0)	89.8 (48.2)
English	511 (366)	0.487 (0.455)	1.576 (1.438)	248.9 (159.4)	805.7 (510.0)

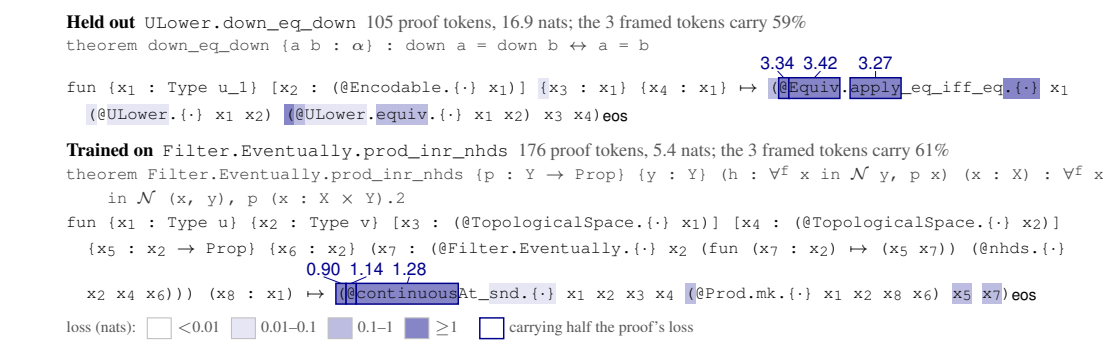

Figure 3: In both proofs, most of the loss sits where the proof names its premise. One held-out and one training proof under the Qwen3-4B term warm start: each token is shaded by loss, with the dearest tokens carrying at least half the proof loss framed and labelled in nats. Both are typical by concentration; selection rule in Appendix F.

4.1 Concentrated loss

Per token, a few term tokens carry nearly all the loss. For the Qwen3-4B warm starts, the top 1% of tokens carry 98.6% of term proof loss, compared with 10.7% for surface Lean and 14.3% for English; the number of tokens needed to account for half the proof loss is 4.0 for terms, 2.0 for surface Lean and 13.0 for English, using the within-proof count illustrated by the frames in Fig. 3. Thus, the pooled 1% result describes concentration relative to sequence length. The nats per token column in Table 2 gives the mean cost of the proof token, pooled over the held-out theorems. For terms, the bars concentrate at low losses and the line at high losses, with 88.8% of proof tokens costing at most 10^{-6} nats. Surface Lean and English spread their tokens across many bins.

Name pieces carry the largest combined share of proof loss (Table F.1). Both proofs in Fig. 3 pay heavily where they name a premise: at the `@` that marks a `Const` token and at the first name pieces. The difficulty lies in choosing which premise to name, especially the ones never cited in training proofs.

Per proof, terms still cost more than surface Lean. At the Qwen3-4B warm starts, terms still cost more than surface Lean per proof, with term-to-surface loss ratios of 2.475 for proofs and 1.081 for statements (Appendix G). The nats per proof columns of Table 2 give the summed loss over each proof block averaged over held-out theorems (Section 3.3). To see what makes the loss gap between statements and proofs, we group held-out theorems by their surface proof: no tactic block, a block with plain tactics only, or a block calling automation such as `simp`. The results show automation has the largest proof ratio; plain tactics and no tactic block are similar, while the from-scratch controls have larger ratios (Table G.1). The larger proof gap accompanies longer explicit proofs, without a corresponding increase in loss per token.

4.2 Teacher-forced loss bounds checked generation from below

From teacher-forced loss alone in this low-entropy regime, the probability of reproducing each reference proof gives a lower bound on generation success that sits just below measured success for

Table 3: Bound@k is Eq. (6) averaged over the sampled theorems. For the term model, the bound computed from teacher-forced loss alone sits just below measured success; surface Lean succeeds more often, also beyond the templates. The 6 templates (`rfl`, `decide` and the like) run without a model on the 1,115 theorems both checkers judge.

model	bound@1 (%)	pass@1 (%)	bound@8 (%)	pass@8 (%) [95% CI]	pass@8 beyond templates (%)
term, Qwen3-4B	10.7	11.9	15.5	19.0 [14.6, 23.8]	5.4
surface Lean, Qwen3-4B	14.1	17.8	21.7	32.0 [27.4, 36.3]	17.6
templates only	—	—	—	15.4 [11.4, 19.7]	—

Table 4: Every warm start ends far below Qwen3-0.6B trained from scratch. Each row is one run using the shared training recipe. Appendix I gives the random-network controls.

		proof loss (nats per theorem)	
initialisation	parameters (M)	before term training	after term training
Qwen3-0.6B	453	12,515	79.3
Qwen3-1.7B	1,435	30,471	72.1
Qwen3-4B	3,665	23,850	74.5
from scratch, Qwen3-0.6B	453	69,694	291.7

the Qwen3-4B term warm start (Table 3). For k independent samples from the full model distribution at unit temperature, the probability of at least one accepted proof satisfies:

$$\text{pass@}k(x) \;\geq\; 1-(1-q_x)^k, \qquad q_x = \exp[-\ell_{\text{proof}|\text{stmt}}(x)]. \tag{6}$$

Reproducing a reference that fits the budget and passes the checker is one way to succeed (Appendix H). For the Qwen3-4B English warm start, the bound is essentially zero because a free-text proof has many valid wordings and the median reference costs 159 nats.

The Qwen3-4B term warm start reaches pass@8 of 19.0% against the 15.5% bound (Table 3) when we sample complete proofs from held-out statements, check terms with the kernel and compile surface Lean at the theorem's original position.

Surface Lean proves more beyond templates. Beyond the templates, the Qwen3-4B surface warm start proves more theorems than the term warm start in 8 samples (Table 3). On the automation group, excluding outputs that call automation brings its rate close to the term rate. On our miniF2F evaluation (Zheng et al., 2022), all term successes in 8 samples are also solved by the templates, while surface Lean proves additional problems (Appendix H).

4.3 Pretrained weights transfer to the term vocabulary

To isolate initialisation, we train the Qwen3-0.6B architecture from random initialisation with the warm start's term tokenisation, training sequence and recipe. The proof loss of the Qwen3-0.6B warm start is lower by a factor of 3.68 than that of Qwen3-0.6B trained from scratch (95% bootstrap interval [3.50, 3.85]). The share of held-out theorems with lower proof loss under the warm start is 99.2%. Before term training, every warm start already sits far below a randomly initialised network of Qwen3-0.6B's shape, whose loss is about a uniform predictor's; all of them improve with training (Table 4). The transferred embeddings alone do not lower a random network's loss, so the head start comes from the pretrained transformer body (Appendix I).

4.4 Term exponents are largest on the measured grid

For models trained from scratch, the fits over the full grid give terms the largest model-size and data-size exponents (Figs. 4 and 5). Among the published fits in Fig. 5, term α has the largest point estimate and term β is level with the largest. These library results agree in direction with the steeper scaling on more compressible synthetic grammars reported by Pandey (2024), though corpora and training setups differ.

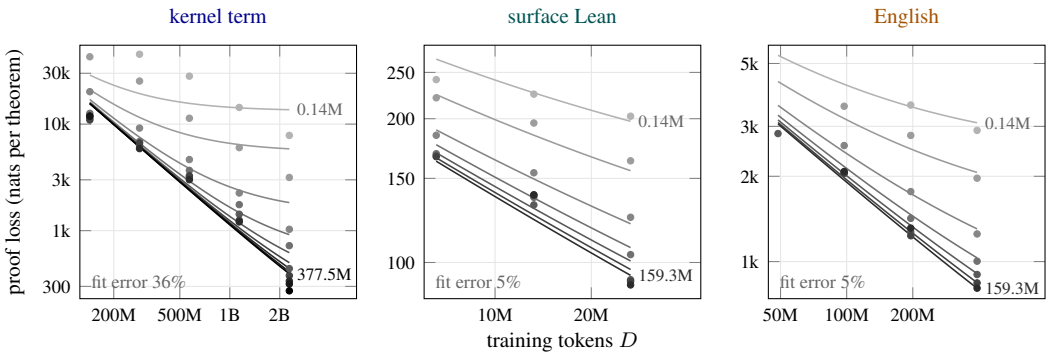

Figure 4: Term loss falls most steeply with data. The fitted law tracks surface Lean and English but misses the steepening decline of term loss. The tag in each panel gives the relative fit error. Markers are measurements and curves are the fit of Eq. (5); darker curves are larger models, labels give the smallest and largest non-embedding parameter counts, and the fits use total parameters.

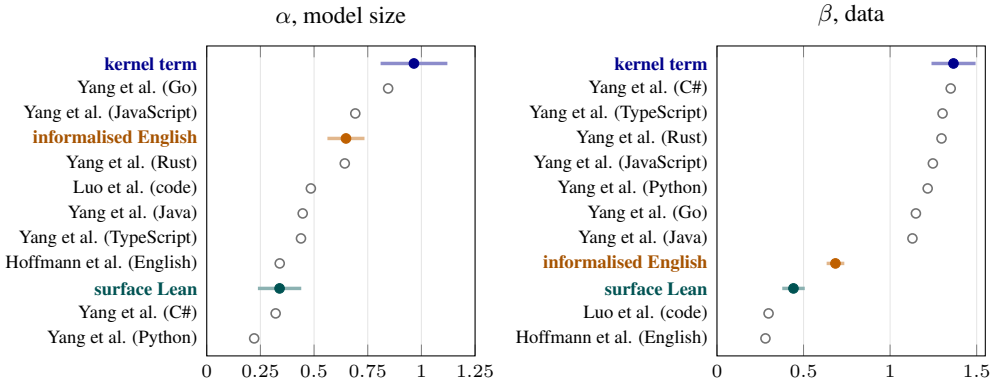

Figure 5: Terms have the largest fitted exponents of our three languages. Fitted exponents, steepest first: α (left) and β (right) for the three languages as posterior medians with 90% intervals (filled), among the 9 published Chinchilla-form fits (open) of Hoffmann et al. (2022), Luo et al. (2026) and Yang et al. (2026), which are point estimates. Our English is informalised (Section 3.1), not the general English of Hoffmann et al. (2022).

We retain the established Chinchilla form, but the term fit is the loosest, with 36% relative fit error against 5% for surface Lean and 5% for English (Fig. 4). For reference, refits in Appendix K that remove exponent priors, exclude embeddings or vary the floor preserve the full-grid ordering. When the grids are restricted to their shared tokens-per-parameter band or epoch window, the comparison between the term and English model-size exponents remains unresolved (Table K.1).

5 Discussion and Conclusion

Under reference prefixes, the post-trained Qwen3-4B term warm start concentrates half of a proof's loss in a few tokens relative to proof length. Adaptive computation or branching could focus effort on uncertain pieces. The concentration measured here uses reference prefixes; a generator would have to estimate uncertainty from its own continuations. Shorter tokenisations with lossless back-references to repeated subterms could fit more proofs into the context, provided they retain a deterministic decoder back to kernel terms.

The post-trained Qwen3-0.6B model transfers from text to the lambda calculus with a new vocabulary, ending with term proof loss lower by a factor of 3.68 than the same network trained from scratch on the same tokens (Section 4.3). We suggest starting from pretrained language models.

From scratch, term proof loss falls more steeply with model size and data than surface Lean's, though the grid's single learning rate slightly inflates the term model-size exponent (Appendix E). If the fits continue with fresh data and model size taken to infinity, term proof loss falls below surface Lean's after about 7 times the current pool of distinct proofs. Expert iteration or autoformalisation could enlarge the pool of distinct proofs.

In the low-loss regime, proof loss predicts checked generation for the post-trained Qwen3-4B term warm start on held-out theorems under the reference-length cap. Under independent unit-temperature sampling, reference-proof probability gives a lower bound on pass@k that sits just below measured success (Table 3). Further reductions in a proof's loss raise its bound in Eq. (6). For generation, checked success per unit of compute is the relevant test.

Taken together, these results support our position: scale pretrained models and the pool of distinct proofs until the kernel's own language becomes a practical medium for generating verified proofs.

Limitations. The scaling results confound data with optimiser updates and depend on the learning rate (Appendix E); generation uses one Mathlib revision, a reference-length cap and inherited Qwen3 pretraining, with term successes on miniF2F limited to kernel computation. The post-trained Qwen3 models may have seen public Mathlib source before transfer. The fitted model-size comparison also depends on the training range. We use one library revision to compare the same theorems in all three languages and compile surface proofs at their original positions. We randomly select the held-out modules, so some proof ideas may already appear in the training dataset; this sampling strategy is standard practice in natural language settings.

AI USE STATEMENT

We used Claude Code to review and polish the content and implement the codebase, including AlphaEvolve-like algorithms (Novikov et al., 2025) that refined our initial ideas to aid formulation. Claude Code also helped check the information-theoretic claims in Section 3.3 and draft this paper. Additional uses included preparing figures and tables, drafting and editing text, searching and summarising literature. Codex was used to review and edit the paper for clarity. The authors reviewed the measurements and claims and are responsible for reviewing the final manuscript and all of its content, including AI-assisted text, claims, and artefacts. As methodology, Qwen3.8-27B generated the informalised English corpus (Appendix A).

REPRODUCIBILITY STATEMENT

The experiments use one Mathlib revision, three languages and a single module-level train–test split (Section 3.1). The code, stored per-declaration scores and generation records will allow the fits, token analyses, reference-probability bounds and acceptance rates to be recomputed without retraining. The 6 generation templates were fixed before the run (Appendix H). The rule selecting Fig. 3 was fixed before any row was drawn (Appendix F). The informalised English prompt was fixed before scoring (Appendix A). Fits and priors are described in Appendix K. Trained checkpoints are not included in the submission. Source code, the population file, the split and the per-declaration held-out scores will be released upon acceptance.

REFERENCES

Andrej Bauer, Matej Petković, and Ljupčo Todorovski. MLFMF: Data sets for machine learning for mathematical formalization. In *Advances in Neural Information Processing Systems*, 2023.

Tamay Besiroglu, Ege Erdil, Matthew Barnett, and Josh You. Chinchilla scaling: A replication attempt. arXiv:2404.10102, 2024.

Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. Graph2Tac: Online representation learning of formal math concepts. In *International Conference on Machine Learning*, 2024.

Mario Carneiro. The type theory of Lean. Master's thesis, Carnegie Mellon University, 2019.

Casey Casalnuovo, Kenji Sagae, and Prem Devanbu. Studying the difference between natural and programming language corpora. *Empirical Software Engineering*, 2019.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

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

Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *International Conference on Automated Deduction*, 2021.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *International Conference on Learning Representations*, 2024.

Konstantin Dobler and Gerard de Melo. FOCUS: Effective embedding initialization for monolingual specialization of multilingual models. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

João Marcos Flach, Álvaro F. Moreira, and Luis C. Lamb. Towards a neural lambda calculus: Neurosymbolic AI applied to the foundations of functional programming. *Journal of the Brazilian Computer Society*, 2026.

Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, and Paolo Torroni. Fast vocabulary transfer for language model compression. In *Conference on Empirical Methods in Natural Language Processing: Industry Track*, 2022.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

Vincent J. Hellendoorn, Premkumar T. Devanbu, and Mohammad Amin Alipour. On the naturalness of proofs. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering: New Ideas and Emerging Results*, 2018.

Tom Henighan, Jared Kaplan, et al. Scaling laws for autoregressive generative modeling. arXiv:2010.14701, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, 2022.

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

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A learning environment for theorem proving. In *International Conference on Learning Representations*, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, et al. Scaling laws for neural language models. arXiv:2001.08361, 2020.

Konstantinos Kogkalidis, Orestis Melkonian, and Jean-Philippe Bernardy. Learning structure-aware representations of dependent types. In *Advances in Neural Information Processing Systems*, 2024.

Mitsuru Kusumoto, Keisuke Yahata, and Masahiro Sakai. Automated theorem proving in intuitionistic propositional logic by deep reinforcement learning. arXiv:1811.00796, 2018.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020.

Guillaume Lample, Timothée Lacroix, Marie-Anne Lachaux, et al. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, 2022.

Yong Lin, Shange Tang, Bohan Lyu, et al. Goedel-Prover: A frontier model for open-source automated theorem proving. In *Conference on Language Modeling*, 2025.

Xianzhen Luo, Wenzhen Zheng, Qingfu Zhu, Rongyi Zhang, Houyi Li, Siming Huang, YuanTao Fan, and Wanxiang Che. Scaling laws for code: A more data-hungry regime. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2022.

Brando Miranda, Avi Shinnar, Vasily Pestun, and Barry Trager. Transformer models for type inference in the simply typed lambda calculus: A case study in deep learning for code. arXiv:2304.10500, 2023.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, et al. Scaling data-constrained language models. In *Advances in Neural Information Processing Systems*, 2023.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. Type-constrained code generation with language models. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2025.

Pengyu Nie, Karl Palmskog, Junyi Jessy Li, and Milos Gligoric. Deep generation of Coq lemma names using elaborated terms. In *International Joint Conference on Automated Reasoning*, 2020.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv:2506.13131, 2025.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. miniF2F-Lean revisited: Reviewing limitations and charting a path forward. In *Advances in Neural Information Processing Systems*, 2025.

Joseph Palermo, Johnny Ye, and Jesse Michael Han. Synthetic proof term data augmentation for theorem proving with language models. Abstract, 7th Conference on Artificial Intelligence and Theorem Proving (AITP), 2022.

Rohan Pandey. gzip predicts data-dependent scaling laws. arXiv:2405.16684, 2024.

Tim Pearce and Jinyeop Song. Reconciling kaplan and chinchilla scaling laws. *Transactions on Machine Learning Research*, 2024.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022.

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

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. In *Advances in Neural Information Processing Systems*, 2024.

Markus N. Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *International Conference on Learning Representations*, 2021.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.

Taro Sekiyama and Kohei Suenaga. Automated proof synthesis for the minimal propositional logic with deep neural networks. In *Asian Symposium on Programming Languages and Systems*, 2018.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In *Advances in Neural Information Processing Systems*, 2024.

The mathlib Community. The Lean Mathematical Library. In *ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2020.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *Advances in Neural Information Processing Systems*, 2025.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989.

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

Huajian Xin, Z. Z. Ren, Junxiao Song, et al. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *International Conference on Learning Representations*, 2025.

Qiyuan Xu, Renxi Wang, Peixin Wang, Haonan Li, and Conrad Watt. A minimalist proof language for neural theorem proving over Isabelle/HOL. *Proceedings of the ACM on Programming Languages*, 2026.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. arXiv:2505.09388, 2025.

Jian Yang, Shuyue Guo, Linzheng Chai, Wei Zhang, Aishan Liu, Chuan Hao, Zhoujun Li, Xin Zhao, Xianglong Liu, Weifeng Lv, and Bryan Dai. Scaling laws for code: Every programming language matters. In *Findings of the Association for Computational Linguistics*, 2026.

Kaiyu Yang, Aidan Swope, Alex Gu, et al. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, 2023.

Po-Hung Yeh and Luke Ong. Most of a formal proof is forced. In *AI for Math Workshop at the International Conference on Machine Learning*, 2026.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

Table R: Appendix roadmap.

appendix	contents	supports
Appendix A	The English prompt	Section 3.1
Appendix B	Expression constructors	Section 3.2
Appendix C	Serialisation grammar	Section 3.2
Appendix D	Loss definitions and proofs	Section 3.3
Appendix E	Training recipe and runs	Section 3.4
Appendix F	Token-loss concentration	Section 4.1
Appendix G	Statement and proof losses	Section 4.1
Appendix H	Sampling and verification	Section 4.2
Appendix I	Transfer controls	Section 4.3
Appendix J	Model-size curves and grids	Section 4.4
Appendix K	Fits and sensitivity checks	Section 4.4
Appendix L	Conversion to nats per token	Section 3.4

A THE ENGLISH PROMPT

This section documents the English prompt used in Section 3.1.

The English prompt was fixed before anything was scored. It gives the model a declaration's surface Lean source and enclosing scope, and asks for a mathematician's statement of the theorem followed

by its proof. It forbids mention of Lean, Mathlib, tactics or identifiers in the source. The parsed statement and proof fields enter the common row format in Appendix D.1. This gives all three languages the statement–proof boundary used in Section 3.3.

The outline and the full write-up use the same prompt with one instruction changed (Listing 3).

Listing 1: This system message is used for both the outline and the full write-up.

```
You are a mathematician writing for a graduate textbook. You are given
    the Lean 4 source of one declaration from Mathlib. Render it in
    English.

Hard constraints, which matter more than style:
- Never mention Lean, Mathlib, tactics, or any identifier that appears
    in the source. No dotted names, no `simp`, no `exact`. A reader must
    not be able to tell the source was formal.
- Use standard mathematical English and LaTeX for symbols.
- Do not restate or quote the Lean code.
- Reply in exactly this form, and write nothing outside it:
<statement>
...
</statement>
<proof>
...
</proof>
```

Listing 2: The user message for the outline. Square brackets stand for each declaration's own text, and the scope block appears only when the declaration has binders from enclosing sections.

```
<scope>
The declaration below is written inside these open sections. The binders
    listed here are in scope for it and constrain the types it mentions,
    even though they do not appear in the source itself.
[the variable and universe binders of the enclosing sections, verbatim]
</scope>

<source>
[the Lean source of the declaration, statement and proof, verbatim]
</source>

"statement": state the theorem as a mathematician would state it in a
    paper -- named hypotheses, standard notation, no formalisation
    artefacts.

"proof": give the idea of the proof in two to four sentences -- the
    strategy, the key lemma or construction it turns on, and why that
    suffices. Do not walk through the steps.
```

Listing 3: This proof instruction replaces the last paragraph of Listing 2 for the full write-up.

```
"proof": give a complete proof at the level of detail of a careful
    textbook -- every step justified, every case distinguished, every
    hypothesis used where it is used. A reader should be able to follow
    it without seeing the source.
```

B LEAN EXPRESSION CONSTRUCTORS

This section describes the expression constructors used by the serialisation in Section 3.2 and identifies the fields it omits.

Constructors and metadata. Lean 4's kernel terms inhabit one inductive type, `Lean.Expr`, with twelve constructors. The twelfth, *metadata* (`mdata`), is a transparent annotation on a subterm that the kernel ignores. The extractor unwraps it before anything is written, so the stream carries eleven node kinds.

Variables, sorts and constants. *Bound variables* (`bvar`) are de Bruijn indices counting binders outward. *Free variables* (`fvar`) name the locals of a context. *Metavariables* (`mvar`) are elaboration holes. *Sorts* (`sort`) carry a universe level, with Prop the sort at level zero. *Constants* (`const`) name environment declarations together with an explicit universe instantiation.

Applications and binders. *Applications* (`app`) are binary and curried. *Lambda* (`lam`) and *dependent function* (`forallE`) abstractions carry a binder name, a domain, a body, and binder-information determining implicitness. *Let* bindings (`letE`) carry a binder name, a type, a value, a body and a non-dependence flag that the kernel does not check. The serialisation drops the binder names and that flag.

Literals and projections. *Literals* (`lit`) are natural-number or string constants. *Projections* (`proj`) select a field of a structure by index.

The propositions-as-types correspondence (Curry, 1934; Howard, 1980) is why theorems and definitions share the `Lean.Expr` type.

C THE SERIALISATION GRAMMAR

This section gives the grammar used in Section 3.2 and the application-order measurements that support the token-loss analysis in Section 4.1.

Reading the stream. Table C.1 lists what the serialiser emits for each node kind, in the order the tokens appear. Each constructor token specifies how many subtrees follow. Variable-length names, numbers and strings end with `<end>`, so a decoder can reconstruct the tree in one left-to-right pass.

Names and numbers. A name's pieces are its components split at dots, underscores and camel-case boundaries, with the dot and the underscore kept as atoms. A piece can be purely numeric, and the reader knows a name has ended only by the `<end>` that closes it. The universe count of a constant follows that `<end>` as one atom. A de Bruijn index and a natural-number literal are their hexadecimal digits, one atom each. Universe levels are written with the level constructors, a level parameter as a name. The count k of a constant's universe arguments and the field index i of a projection are decimal atoms without a terminator. The name of a level parameter after `<L_param>` is split into pieces and closed by `<end>` like any other name.

Universe levels and binder fields. A universe level u uses `<L_zero>`, `<L_succ>` u, `<L_max>` u v, `<L_imax>` u v, or `<L_param>` followed by a name. Binder information κ is a token.

Application order. An application $f\ a_1 \dots a_k$ is emitted as k consecutive `App` tokens and only then the head f, so a left-to-right model fixes the arity before it names what is applied. Table C.2 gives the losses for the from-scratch model `N169_D2821_term`, with 168.8M parameters. Its mean loss over all proof tokens is 0.0451 nats per token.

For this from-scratch model, the first piece of the head's name carries more loss than all the `App` tokens together. An n-ary application token or a head-first order is untested. At the Qwen3-4B term warm start, the `App` tokens carry more loss than the head's first name piece, with shares of 15.53% and 13.17%, respectively.

Table C.1: The serialisation, one line per constructor, as the model reads it. The c_j and S_j are name pieces, the h_j are hexadecimal digits, and κ is binder information. Free variables and metavariables do not occur in a closed declaration.

constructor	emitted tokens	then
`bvar` i	`<bvar>` $h_1 \dots h_j$ `<end>`	—
`fvar` x	`fvar:`x	—
`mvar` m	`mvar:`m	—
`sort` u	`Sort` u	—
`const` $c\, u_1 \dots u_k$	`Const` $c_1 \dots c_m$ `<end>` $k\, u_1 \dots u_k$	—
`app` $f\, a$	`App`	f, then a
`lam` $(x : t), b$	`Lam <bi_`κ`>`	t, then b
`forallE` $(x : t), b$	`Pi <bi_`κ`>`	t, then b
`letE` $x : t := v; b$	`Let`	t, v, then b
`lit` (natural n)	`<natlit>` $h_1 \dots h_j$ `<end>`	—
`lit` (string s)	`<strlit>` $h_1 \dots h_j$ `<end>`	—
`proj` $S\, i\, e$	`Proj` $S_1 \dots S_m$ `<end>` i	e
`mdata` e	(unwrapped)	e

Table C.2: Application prefixes and head names in the from-scratch model. Shares are over scored proof tokens and their total loss.

tokens	token share (%)	loss share (%)	nats/token
`App`	12.56	12.53	0.0450
first piece of the head's name	4.42	13.23	0.1351

D Information-theoretic quantities

This section gives the definitions, the physical intuition and proofs behind the loss measurements in Section 3.3. It proves the statement–proof split and states what elaboration implies for entropy.

Loss as message length. Imagine sending a theorem to someone who shares the model π as a codebook, so likely text needs a shorter message. The loss $\ell(x) = -\log \pi(x)$ is that ideal message length in nats, and $e^{-\ell(x)}$ is the probability of sampling the exact text at unit temperature. The cross-entropy $\mathrm{CE}(p, \pi) = \mathbb{E}_{x\sim p}[\ell(x)]$ averages this length over a source p.

Entropy and extra length. Entropy measures how many possibilities the source leaves open, as in Boltzmann's count of equally likely states. For W equally likely versions, $H(p) = \log W$ nats; more generally, $e^{H(p)}$ is the effective number of equally likely versions. A codebook matched to the source attains the minimum average length $H(p) = \mathrm{CE}(p, p)$, while the Kullback–Leibler divergence $\mathrm{KL}(p \,\|\, \pi) \geq 0$ is the extra length paid when the model's probabilities differ from the source's (Cover & Thomas, 2006). Thus, population cross-entropy is an upper bound on source entropy; a finite held-out mean estimates that cross-entropy.

D.1 Rows and blocks

For a language $c \in \{\text{term}, \text{surface}, \text{English}\}$, let p_c be the source of versions X^c, split into X^c_{stmt} and X^c_{proof}. The encoded row $X = \iota_c(X^c)$ begins with `<bos>`, has a statement block ending at the unique `<sep_type>` and has a proof block ending at the unique `<eos>`. We write these row blocks as X_{stmt} and X_{proof}, and `<bos>` is given rather than scored. The tokenisation is injective up to the omitted fields described in Appendix B, so distinct retained versions give distinct rows.

An autoregressive model π assigns a row its product of next-token probabilities. The whole-theorem loss ℓ, statement loss ℓ_{stmt} and proof loss $\ell_{\mathrm{proof|stmt}}$ are defined in Section 3.3. For every autoregressive model and every row,

$$\ell(x) = \ell_{\mathrm{stmt}}(x) + \ell_{\mathrm{proof|stmt}}(x).$$

Taking expectations over any source preserves this split,

$$\mathrm{CE}(p_c, \pi) = \mathrm{CE}_{\mathrm{stmt}}(p_c, \pi) + \mathrm{CE}_{\mathrm{proof|stmt}}(p_c, \pi), \tag{7}$$

which is Eq. (4).

Loss and entropy. Each loss is the negative log-probability of its row or block, with proof-block probability conditioned on the statement block. Statement-block probabilities sum to at most one because the unique final <sep_type> makes distinct statement blocks disjoint sampling events, and proof-block probabilities conditional on a statement do the same because each proof ends at the unique <eos>. Hence the mean losses obey

$$\mathrm{CE}_{\mathrm{stmt}}(p_c, \pi) \geq H(X_{\mathrm{stmt}}), \qquad \mathrm{CE}_{\mathrm{proof|stmt}}(p_c, \pi) \geq H(X_{\mathrm{proof}} \mid X_{\mathrm{stmt}}),$$
$$\mathrm{CE}(p_c, \pi) \geq H(X),$$

with equality exactly when the model's block probabilities match the source's corresponding distributions. The proof loss charges <eos> because without that stop token a proof and a longer proof extending it would overlap as sampling events.

Data processing inequality. A deterministic map cannot add entropy, since $H(Y) = H(f(Y)) + H(Y \mid f(Y)) \geq H(f(Y))$. If the term version is a function of the surface Lean version and its statement is a function of the surface statement, then

$$H(X^{\text{term}}) \leq H(X^{\text{surface}}), \qquad H(X^{\text{term}}_{\mathrm{stmt}}) \leq H(X^{\text{surface}}_{\mathrm{stmt}}).$$

The injective tokenisation carries these comparisons to the rows and statement blocks that the models score, up to the omitted fields.

Elaboration also reads the module's context, so the functional assumption requires that context to be fixed or supplied. The entropy comparison concerns sources, while finite models may order their mean losses either way. It does not order $H(X^{\text{term}}_{\mathrm{proof}} \mid X^{\text{term}}_{\mathrm{stmt}})$ against $H(X^{\text{surface}}_{\mathrm{proof}} \mid X^{\text{surface}}_{\mathrm{stmt}})$, and every scaling law in this paper is fitted on proof loss given the statement. An English version is sampled instead of computed by elaboration, so this comparison says nothing about English.

E Training configuration

This section gives the training recipe, architectures and token budgets, including warm starts and a probe of how the learning rate and update count change the losses.

Grids. Each from-scratch cell is a decoder-only transformer trained at one model size and token budget (Table E.1). The total parameter count N at a given width varies across languages because their embedding vocabularies differ (Table E.3). Each token budget is trained as an independent run with its own schedule. Each language's token budgets reach at most 2 epochs of its training theorems (Table E.4). Training sequences may pack several theorem rows, but the attention mask prevents attention across rows in all three languages.

Warm starts. A warm start differs from a from-scratch model in its initialisation and in the transformer shape inherited from its Qwen3 model. Among the three term warm starts, the Qwen3 checkpoint and the shape it fixes are the only intended differences.

Qwen3-0.6B trained from scratch has the same architecture, uses the same order of training steps and the same tokenisation. It uses the term Chinchilla fit.

Vocabulary transfer initialises the tied input embedding and output head together. A small perturbation separates identical embedding rows. The transformer body is copied weight-for-weight, so the three Qwen3 models do not represent three widths of a single architecture.

The surface Lean and English warm starts use Qwen3-4B, the same transfer procedure and their own byte-level BPE vocabularies.

Recipe probe. The recipe was fixed without tuning. A probe of 12 runs outside the grid measures the cost of that choice (Table E.2). Each repeats a grid model with one setting changed; none enters a fit or a figure. A change in log loss smaller than 0.06 counts as no change. This band is just under two standard deviations of log loss across seeds.

Table E.1: Training recipe for the grids and warm starts. Warm starts use their language's largest token budget.

setting	from scratch	warm starts
optimiser	AdamW	
betas	$\beta_1 = 0.9$, $\beta_2 = 0.95$	
weight decay	0.1	
peak learning rate	3×10^{-4}	
warm-up rule	5% of the run's update budget, minimum 3 steps, maximum 20	
schedule	cosine decay to 0.1 of the peak over the run's update budget	
precision	bfloat16 computation; fp32 master weights	
context	65,536 tokens	
embedding and head	tied	
initialisation	standard deviation 0.02	Qwen3 body and transferred embeddings
transformer shape	grid (Table E.3)	inherited from the Qwen3 model
attention	head width 128; query–key normalisation	inherited from the Qwen3 model
Nominal tokens per update		
term	2,097,152	
surface Lean	131,072	
English	1,048,576	

Learning rate. Raising the peak learning rate from 3×10^{-4} to 1×10^{-3} lowers term proof loss at every probed width. The reductions range from 26.4% to 56.6%. The widest model gains least, so loss falls more slowly with model size at the higher rate. At the largest token budget, the negative log-log slope of proof loss against total parameters N, read from the two end widths, is 0.711 at the grid's rate and 0.634 at the higher rate (Table E.2).

Both slope differences exceed the threshold of 0.05 fixed before the runs. The end-point difference has a standard error of about 0.015. This estimate uses the seed spread of replicated term models because every probe model was trained once. The term model-size exponent is therefore inflated by the grid's single learning rate over the probed widths (Fig. 5). We expect the true exponent to be in the lower half of its posterior interval.

Update count. On surface Lean, processing the same tokens in 8 times as many updates, each an eighth the size, lowers loss by 44.9% to 60.2%. The absolute losses, fitted amplitudes and fitted floors are therefore properties of a recipe with few updates at one rate.

Table E.2: The recipe probe. *Grid* and *probe* give proof loss in nats per theorem for the original and modified runs; *ratio* is the second over the first. The term rows are at the largest token budget; surface Lean token counts are tokens processed.

language	d_{model}	change	grid	probe	ratio
term	64	rate 1×10^{-3}	7,834.7	4,222.9	0.539
term	64	rate 3×10^{-3}	7,834.7	3,448.1	0.440
term	64	rate 1×10^{-2}	7,834.7	3,430.4	0.438
term	128	rate 1×10^{-3}	3,157.8	1,368.9	0.434
term	128	rate 3×10^{-3}	3,157.8	1,379.0	0.437
term	256	rate 1×10^{-3}	1,030.3	546.8	0.531
term	512	rate 1×10^{-3}	440.5	324.3	0.736
surface Lean	512	8× updates, 6.5M tokens	167.4	83.4	0.498
surface Lean	512	8× updates, 13.2M tokens	132.0	60.7	0.460
surface Lean	512	8× updates, 26.5M tokens	92.0	50.7	0.551
surface Lean	768	8× updates, 13.2M tokens	138.4	55.1	0.398
surface Lean	768	8× updates, 26.5M tokens	89.8	47.2	0.526

Table E.3: The nine architectures of the grid and their total parameter count N on each language. The seven widths up to 768 are trained on all three languages and the two largest on terms only. Qwen3-0.6B trained from scratch has the same shape as the warm-start (Appendix I).

						N (M)		
d_{model}	layers	heads	kv heads	d_{ff}	non-embedding (M)	term	surface Lean	English
64	2	1	1	192	0.14	0.9	0.3	1.2
128	3	2	1	384	0.74	2.3	1.0	2.8
256	6	4	2	768	5.9	9.1	6.4	10.1
384	9	6	3	1152	19.9	24.7	20.7	26.2
512	12	8	4	1536	47.2	53.6	48.3	55.6
640	15	10	5	1920	92.2	100.1	93.5	102.7
768	18	12	6	2304	159.3	168.8	160.9	171.9
896	21	14	7	2688	252.9	264.0	—	—
1024	24	16	8	3072	377.5	390.2	—	—

Table E.4: Token budgets of each grid. Epochs count passes over the training theorems. At the largest term budget, the model count includes Qwen3-0.6B trained from scratch (Appendix I).

language	tokens processed D (M)	updates	epochs	models
term	143.5	84	0.10	7
	286.0	168	0.20	7
	570.8	336	0.40	7
	1,140.1	672	0.81	7
	2,283.2	1,345	1.62	10
surface Lean	6.5	50	0.49	6
	13.2	101	0.99	7
	26.5	203	1.99	7
English	48.7	47	0.24	1
	97.4	94	0.49	6
	195.8	189	0.98	7
	392.5	379	1.97	7

F TOKEN-LEVEL PREDICTION LOSS

This section gives the example-selection rule, the split by premise familiarity and the token-level measurements behind the loss-concentration claims in Section 4.1.

Example-selection rule. The examples in Fig. 3 are scored under reference prefixes and selected by a rule fixed before drawing any example. Eligible theorems have statements of at most 160 characters and proofs of 80–200 tokens. Every framed name must be cited at least 10 times in training proofs, and none may be declared in a held-out module. Among eligible theorems, we select the proof nearest the medians of its population for the number of tokens carrying half the loss, the highest-loss token's share and the name-piece share. The population is all held-out theorems for the held-out example and a uniform draw of training theorems for the training example. The display uses x_k for the k-th binder.

Premise familiarity. For the Qwen3-4B term warm start, constants never cited in training proofs account for 1.44% of held-out constant citations, but their name pieces carry 24.64% of all held-out proof loss, more than half the name-piece loss. Of those citations, 67.6% refer to constants declared in held-out modules. The name pieces of a never-cited constant carry 2.924 nats, against 0.025 for constants cited at least 10 times in training proofs. Never cited in a training proof does not mean never seen: a name can occur in a statement or Qwen3-4B's pretraining text.

Pooled loss and individual tokens. Figure 1's cumulative proof-loss share is shown in Figure F.1 Within a theorem under the Qwen3-4B term warm start, the highest-loss token accounts for a median 20.9% of proof loss (quartiles 11.6%–37.8%).

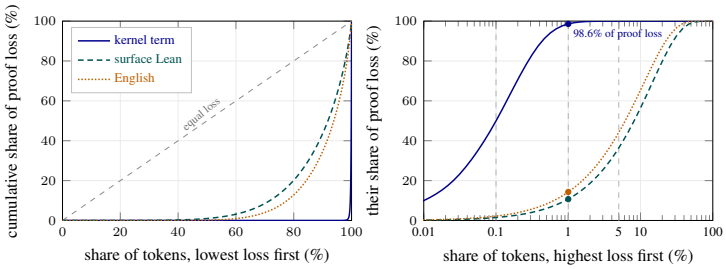

Figure F.1: Proof-token loss for the Qwen3-4B warm starts, one per language. Left: cumulative loss when proof tokens are sorted from lowest to highest loss. Right: the share of total proof loss carried by the highest-loss $q\%$ of tokens.

Table F.1: The proof tokens of the term language for the Qwen3-4B warm start by their job in the serialisation (Appendix C).

token class	share of tokens (%)	share of proof loss (%)	nats per token
constructor tokens	19.35	32.51	0.0168
bound variables	21.22	19.27	0.0091
universe levels and counts	29.80	4.30	0.0014
first piece of a name	5.86	13.89	0.0237
later pieces of a name	17.30	28.86	0.0166
end of a name	5.86	0.73	0.0013
literals	0.61	0.28	0.0046
<eos>	0.01	0.14	0.1073
other	0.00	0.01	0.2149

Correct next-token predictions. For each model, we report the share of tokens whose argmax equals the reference token under teacher forcing. We also report the share of whole-theorem loss CE those tokens carry (Table F.2). Both shares use the whole loss mask, including statements and proofs. A term model's most likely next token is the reference token almost everywhere.

Table F.2: Tokens already predicted under reference prefixes, by language and initialisation.

N (M)	language	argmax = reference token (% tokens)	their share of CE (%)
168.8	term, from scratch	98.47	22.74
160.9	surface Lean, from scratch	48.77	12.98
171.9	English, from scratch	62.39	15.11
3,665.3	term, Qwen3-4B	99.56	17.49
3,638.8	surface Lean, Qwen3-4B	75.84	12.30
3,675.5	English, Qwen3-4B	83.24	19.66

G STATEMENT AND PROOF LOSSES

This section compares statement and proof losses by tactic use, model size and language, supporting Section 4.1. It also gives the extrapolated crossings of the fitted laws and the measurements for every term model at the largest token budget.

G.1 TACTIC USE

Table G.1 splits the held-out theorems by what the surface proof leaves to Lean. A tactic block is the keyword `by` standing alone outside comments. The automation group calls at least one of 43 tactics that search, normalise or decide: `simp`, `simpa`, `simp_all`, `dsimp`, `simp_rw`, `grind`, `omega`, `lia`, `aesop`, `aesop_cat`, `cat_disch`, `norm_num`, `linarith`, `nlinarith`, `positivity`, `decide`, `tauto`, `field_simp`, `ring`, `ring_nf`, `abel`, `module`, `noncomm_ring`, `linear_combination`, `polyrith`, `gcongr`, `fun_prop`, `continuity`, `measurability`, `norm_cast`, `push_cast`, `exact_mod_cast`, `assumption`, `trivial`, `infer_instance`, `bound`, `order`, `finiteness`, `compute_degree`, `monicity`, `mono`, `exact?`, `apply?`.

Bootstrap comparisons. The loss ratios compare mean losses at the matched from-scratch models and Qwen3-4B warm starts (Table G.1). Their 95% intervals come from a bootstrap over the 755 held-out modules, with the same resamples for all three groups and both loss units.

Loss gaps by tactic use. At the Qwen3-4B warm starts the term-to-surface Lean ratio of proof loss per theorem is higher in the automation group than in the group without a tactic block, by 1.07 [0.72, 1.44]. The plain-tactics group has a lower ratio than the group without a tactic block by -0.08 [-0.28, 0.10], with an interval spanning zero.

Table G.1: Term and surface Lean by tactic use. The panels give theorem counts and mean proof lengths, then term-to-surface Lean loss ratios.

		proof tokens		
tactic use	theorems	term	surface Lean	token ratio term/surface Lean
no tactic block	9,393	2,049	16.1	127
plain tactics	3,085	11,159	53.8	207
automation	5,088	15,235	56.6	269

tactic use	from scratch	Qwen3-4B
Proof loss in nats per theorem, term/surface Lean		
no tactic block	2.45 [2.19, 2.78]	1.97 [1.77, 2.26]
plain tactics	3.16 [2.83, 3.56]	1.89 [1.69, 2.15]
automation	4.80 [4.36, 5.31]	3.04 [2.63, 3.60]
Statement loss in nats per theorem, term/surface Lean		
no tactic block	1.42 [1.26, 1.59]	1.16 [1.10, 1.25]
plain tactics	0.97 [0.90, 1.05]	0.97 [0.94, 1.01]
automation	0.95 [0.90, 1.01]	1.02 [0.98, 1.07]
Proof loss in nats per token, term/surface Lean		
no tactic block	0.0193 [0.0173, 0.0213]	0.0155 [0.0131, 0.0184]
plain tactics	0.0152 [0.0138, 0.0170]	0.0091 [0.0082, 0.0103]
automation	0.0178 [0.0162, 0.0198]	0.0113 [0.0096, 0.0136]

G.2 MODEL SIZE

Loss per token across model sizes. We plot two comparisons of statement and proof loss against total parameters N for the 10 from-scratch term models at the largest token budget and the three Qwen3 warm starts (Fig. G.1). The left panel is the share of held-out theorems with $\ell_{\mathrm{stmt}}(x)/n_x > \ell_{\mathrm{proof|stmt}}(x)/m_x$, the statement's loss per token above the proof's. The share rises from 44.24% at 0.9M parameters to 86.04% at 168.8M.

Median proof-to-statement ratio. The right panel Table G.2 is the ratio of the median $\ell_{\mathrm{proof|stmt}}$ to the median ℓ_{stmt}. If that line continues, the median losses reach parity at 925M parameters. The 264.0M model's ratio is 1.46 against a predicted 1.28 [1.10, 1.50]. The 390.2M model's ratio is

1.34 against 1.19 [1.01, 1.39]. Qwen3-0.6B trained from scratch lies above this interval at 1.42; The Qwen3-0.6B warm start lies below it at 0.76.

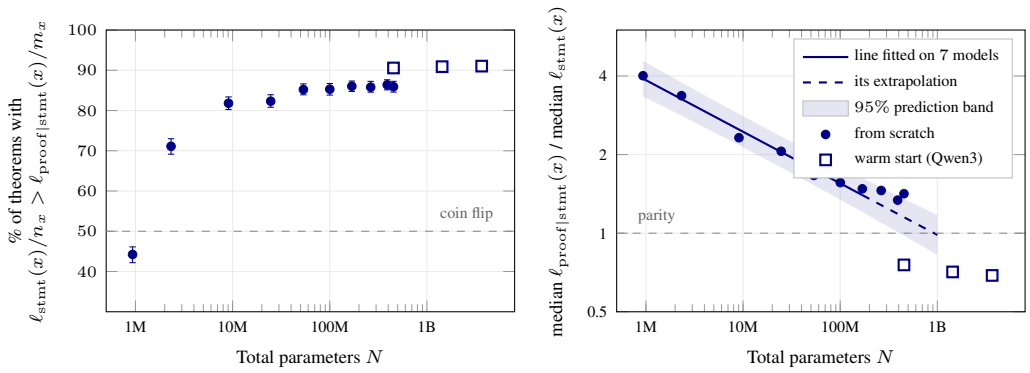

Figure G.1: Statement and proof losses across term models. Left: the share with higher statement loss per token; filled circles are from scratch and hollow squares are Qwen3 warm starts. Right: median proof-to-statement loss ratio, with the line fitted on the seven smallest from-scratch models solid, extrapolation dashed and a 95% prediction band.

Loss distributions. For Qwen3-0.6B trained from scratch, the median proof loss exceeds the median statement loss (100.7 against 71.1 nats, Fig. G.2). For the Qwen3-0.6B warm start, which differs only in initialisation, the median proof loss is below the median statement loss (30.9 against 40.9 nats), so proof loss falls by the larger factor.

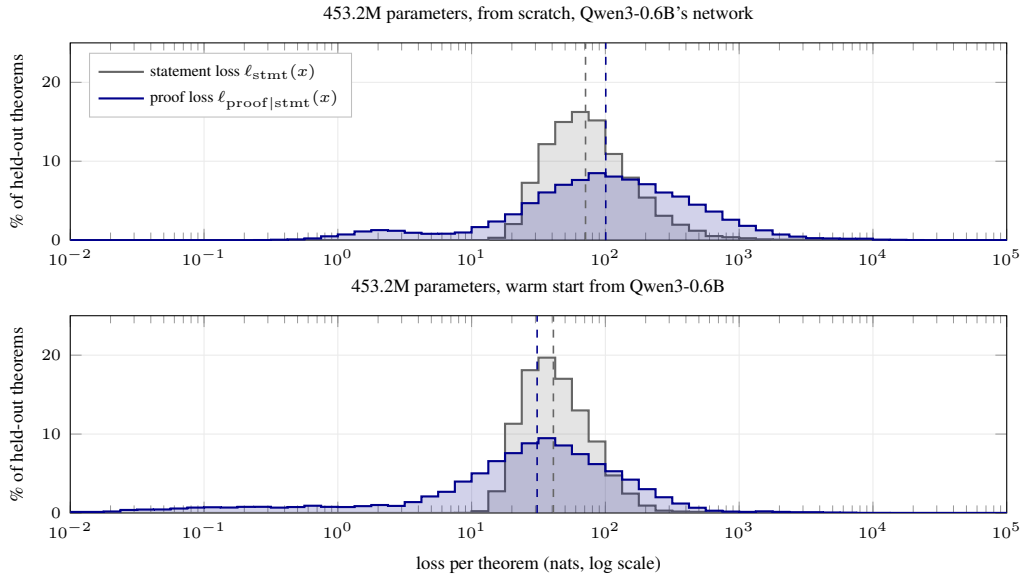

Figure G.2: The statement loss ℓ_{stmt} and the proof loss $\ell_{\mathrm{proof|stmt}}$ of every held-out theorem, in nats on a logarithmic axis. Models are Qwen3-0.6B trained from scratch and the Qwen3-0.6B term warm start, trained on the same tokens. Dashed rules mark the medians.

G.3 Region comparisons and extrapolated crossings

Region comparisons. Table G.3 compares statements, conditional proofs and whole theorems on the 17,566 held-out theorems. The ratio divides each region's term mean loss by its surface Lean mean loss. "term lower loss" is the share of theorems with lower term loss.

Table G.2: Every term model at the largest token budget. Rows below the rule are Qwen3 warm starts; the others are trained from scratch, the last being Qwen3-0.6B trained from scratch. The "outright" column gives the share with $\ell_{\text{stmt}}(x) > \ell_{\text{proof|stmt}}(x)$, and the final two columns give argmax statistics over the whole theorem (Table F.2).

N (M)	median ratio	higher/token (%)	95% interval	outright (%)	argmax reference token (%)	share of CE (%)
0.9	4.01	44.24	[42.19, 46.13]	26.62	71.70	23.26
2.3	3.36	71.11	[69.15, 72.99]	28.29	88.86	22.93
9.1	2.32	81.81	[80.24, 83.38]	33.19	96.25	22.09
24.7	2.06	82.32	[80.82, 83.93]	33.54	97.20	22.30
53.6	1.66	85.22	[83.91, 86.61]	36.72	98.09	23.76
100.1	1.56	85.31	[83.84, 86.72]	37.48	98.36	22.91
168.8	1.48	86.04	[84.75, 87.33]	38.65	98.47	22.74
264.0	1.46	85.81	[84.56, 87.25]	38.79	98.59	21.81
390.2	1.34	86.39	[85.15, 87.64]	40.65	98.75	21.84
453.2	1.42	85.92	[84.58, 87.26]	39.22	98.68	22.33
453.2	0.76	90.57	[89.43, 91.61]	60.71	99.52	21.63
1,434.8	0.71	90.90	[89.74, 91.96]	63.19	99.55	19.49
3,665.3	0.69	91.03	[89.87, 92.05]	63.55	99.56	17.49

Table G.3: Term and surface Lean by region. Crossing brackets are 90% posterior intervals. "None" means the fitted laws do not meet.

	from scratch, matched size		Qwen3-4B warm starts	
region	ratio	term lower loss (%)	ratio	term lower loss (%)
statement, CE_{stmt}	1.182 [1.092, 1.294]	61.19 [57.63, 64.88]	1.081 [1.037, 1.136]	53.58 [50.43, 56.64]
proof, $\text{CE}_{\text{proof\|stmt}}$	3.751 [3.451, 4.095]	10.74 [9.72, 11.95]	2.475 [2.192, 2.869]	20.04 [18.34, 21.56]
whole theorem, CE	2.359 [2.196, 2.550]	24.43 [21.94, 26.97]	1.617 [1.487, 1.797]	30.66 [28.17, 33.34]

	laws meet at k	
region	matched N	$N \to \infty$
statement, CE_{stmt}	1.10	1.03 [0.69, 1.86]
proof, $\text{CE}_{\text{proof\|stmt}}$	none	4.50 [2.83, 8.41]
whole theorem, CE	none	2.90 [1.87, 5.36]

Extrapolated crossings. The lower panel reports where fits of the Chinchilla form meet (Eq. (5)). The term statement fit has relative fit error 31%, against 6% for surface Lean. The whole-theorem fits have relative fit errors of 34% for terms and 6% for surface Lean. The crossings inherit the larger error of the term fits. The multiplier k counts theorems processed relative to the 255,275 processed by the matched term model. That model's training covers 1.62 passes over the 157,686 training theorems. At matched model sizes, only the statement fits cross; dropping the model-size contribution by taking $N \to \infty$ also produces crossings for proofs and whole theorems. The proof crossing with $N \to \infty$, at $k = 4.50$, needs about 7 times as many distinct theorems as the training pool holds. These crossings extrapolate beyond the measured budgets and treat additional theorems processed as fresh data, whereas the training pool repeats after one pass.

Other crossing estimates. The proof fits have no crossing at the largest model sizes, 390.2M parameters for terms and 160.9M for surface Lean (Table E.3). Straight lines through each language's top two token budgets at the matched sizes meet for proofs at $k = 2.66$ if their trends continue with fresh data.

Statement loss excess. Term statements have the higher mean loss, yet the lower loss on most theorems (Table G.3). The theorems with the largest term statement loss excess over surface Lean (the top 1%) carry 66.4% of the summed excess at matched size and 65.7% at the Qwen3-4B warm starts.

H PROOF GENERATION AND VERIFICATION

This section gives the sampling protocol, checker rules, reference-probability bound and generation results supporting Section 4.2.

H.1 SAMPLING AND THE EVALUATION POPULATION

Models and prompts. A model receives a held-out statement in its own language and generates a whole proof. We compare the size-matched from-scratch models with warm starts from the post-trained Qwen3-4B checkpoint, so the models differ in initialisation as well as size. All use the training split and checkpoints described in Appendix E. The comparisons share theorems, while the languages differ in prompt information, training-token counts and optimiser updates. The Qwen3 model may also have encountered public Mathlib source or benchmark problems before this study.

Population and weights. We evaluate on the 14,970 held-out theorems whose reference kernel term is at most 16,384 tokens. We draw 1,118 theorems, stratifying by tactic use (no tactic block, plain tactics or automation) and four reference-term length bands. Each theorem is weighted by its stratum's population size divided by the number drawn. These weights estimate rates for the held-out theorems under the reference cap, and intervals resample whole Mathlib modules.

Decoding settings.

- For each prompt we run one greedy decode and 8 independent temperature-1 samples, with no top-k or top-p truncation.
- On Mathlib, term outputs may use 32,768 new tokens and surface outputs 2,048; every reference under the cap fits its budget.

An output that does not emit `<eos>` within its budget counts as a failure. Sampled pass@1 averages the success indicators of individual samples; pass@8 records whether at least one of the eight succeeds. The reference-probability bound applies to these sampled rates, not to greedy decoding.

H.2 WHAT THE CHECKERS ACCEPT

Kernel terms. Each output is decoded to a Lean expression and checked by the Lean kernel against the statement's type. The axiom guard follows dependencies and allows only `propext`, `Classical.choice` and `Quot.sound`.

Declaration availability. The full Mathlib environment already contains the target theorem. To prevent a generated term from proving a theorem by citing itself, we also check declaration availability. Every cited constant must come from an imported module or an earlier declaration in the same file; generated auxiliaries inherit their parent declaration's position. Cited constants that depend on the target theorem are rejected. There is one explicit exception: a term identical to the Mathlib reference is accepted, including auxiliaries created for that reference during elaboration. This availability check was added after inspecting the Qwen3-4B term warm start's first greedy outputs, then applied to every term run. It rejects 16 outputs from that greedy run which the unrestricted kernel accepted. All 1,118 reference terms pass the final checker.

Audit of the reference exception. Among the drawn references, 194 of 1,118 cite their own elaboration helpers by name, with the helper bodies stored as separate declarations outside the serialised proof. Recounting all four term runs with the same availability rule for every output, including the reference, changes no accepted output or pass rate. The Qwen3-4B pass@8 reference bound on the drawn theorems falls only from 15.527% to 15.516%.

Surface proofs. We rebuild the theorem's Mathlib source file up to the target proof, insert the generated text and compile it in a fresh Lean process. This preserves the local variables, notation and tactic environment and removes declarations after the theorem. Lean may run any tactic named in the output. Acceptance requires successful compilation and the same axiom allowance as for terms, excluding proofs that use `sorry`. The Mathlib reference proofs pass on 1,115 of the 1,118 sampled held-out theorems. The remaining 3 are excluded from surface Lean rates and paired comparisons.

H.3 Reference probability and measured success

Deriving the bound. For a held-out theorem x whose reference proof fits the budget and passes the checker, the chain rule gives $q_x = \prod_{i=n_x+1}^{n_x+m_x} \pi(x_i \mid x_{<i}) = \exp[-\ell_{\text{proof}|\text{stmt}}(x)]$, with <eos> included. The event of sampling x_{proof} is a subset of the event of sampling an accepted proof. Missing the reference proof in k independent unit-temperature draws has probability $(1-q_x)^k$, which gives the bound (Eq. (6)). The lower bound is on the model's expected success probability; a finite sample estimate has sampling uncertainty. We average per-theorem bounds using the same population or stratum weights as the corresponding pass rates.

Bounds and measured rates. For the held-out theorems under the reference cap, bound@1 is 1.1% for the size-matched from-scratch term model and 10.3% for the Qwen3-4B term warm start. Table 3 uses the bounds estimated from the sampled held-out theorems, so the bound and observed rate share the same draw. Table H.1 adds greedy decoding and results by tactic use. Counts of accepted outputs are unweighted sample counts, whereas a pass@8 success counts a theorem once.

Exact matches. The Qwen3-4B term warm start produces 738 accepted samples. Of these, 535 reproduce the reference and 203 differ from it.

Table H.1: Kernel-accepted proofs as a percentage of the 14,970 held-out theorems under the reference cap, overall and by the tactic use of the Mathlib proof. The surface rows include proofs that call Lean's automation; the term rows cannot.

	greedy, one proof (%)				sampled (%)	
model	all	no tactic block	plain tactics	automation	pass@1	pass@8
kernel term, 169M	3.00	4.84	0.00	0.14	1.61	5.27
kernel term, Qwen3-4B	14.94	22.72	1.74	3.28	11.94	19.04
surface Lean, 161M	10.90	17.25	0.27	1.14	7.93	14.03
surface Lean, Qwen3-4B	22.71	31.54	5.15	11.10	17.80	31.98

Alternative proofs. The Qwen3-4B term warm start produces 92 distinct accepted terms that differ from the reference. Of these, 40 use reflexivity and 52 apply another constant at the head of the proof, once its binders are stripped. These are syntactic differences between proof terms; identifying distinct mathematical arguments would require a stronger equivalence criterion.

Templates baseline. On the 1,115 Mathlib theorems both checkers can check, we test what 6 reflexivity and decision templates, fixed before the run, can prove without a model. The templates are `rfl`, `by rfl`, `Iff.rfl`, `by intros; rfl`, `by decide` and `by decide +kernel`. We try every template on every theorem, use the same stratum weights and keep the whole common set in the denominator.

Overlap and success beyond templates. After stratum weighting, the templates prove an estimated 71.5% of the theorems solved by the Qwen3-4B term warm start and 44.9% of those solved by the Qwen3-4B surface Lean warm start. Beyond the templates, term pass@8 is 5.44% [3.48, 7.63], while surface Lean pass@8 is 17.63% [14.31, 21.04].

Recounting surface outputs without automation. On theorems whose Mathlib proof calls automation, retaining only surface outputs that do not call automation gives pass@8 of 4.54% [2.28, 7.18], compared with 4.49% [2.20, 7.17] for terms. This recount was chosen after observing the

outputs. It leaves the surface elaborator in place and does not predict how a model instructed to avoid automation would behave.

Paired outcomes. On the 1,115 theorems both checkers can check, the Qwen3-4B warm starts both succeed on 141. The Qwen3-4B term warm start alone succeeds on 25, and the Qwen3-4B surface Lean warm start alone on 183. The term-only successes add 2.39 percentage points [1.23, 3.77] to the observed surface Lean rate.

H.4 miniF2F beyond the training corpus

Benchmark port. We evaluate the Qwen3-4B term and surface Lean warm starts on 488 miniF2F problems (Zheng et al., 2022). We combine the 244 test and 244 validation problems. The Lean 4 statements distributed with DeepSeek-Prover-V1.5 (Xin et al., 2025) are ported to the study's Mathlib revision. The port updates notation, and separately corrects 22 statements that admit compiled counterexamples as distributed, following the issues also identified by benchmark audits (Ospanov et al., 2025). We report the 466 untouched statements separately so the effect of these corrections is visible. Both models receive only the statement in their own language, use a 32,768-token output budget on this benchmark, and receive the same greedy and sampled evaluations. Intervals resample problems, rather than Mathlib modules.

Table H.2: Accepted proofs as a percentage of miniF2F problems, on all 488 statements with 22 corrected. A dash marks a greedy run, which has one sample only.

		pass@1			pass@8	
model	decoding	all	test	validation	all	test
kernel term, Qwen3-4B	greedy	0.20	0.41	0.00	—	—
kernel term, Qwen3-4B	sampled	1.38	1.28	1.49	5.94	6.15
surface Lean, Qwen3-4B	greedy	3.48	3.69	3.28	—	—
surface Lean, Qwen3-4B	sampled	4.46	4.76	4.15	17.21	18.44

In eight samples, the Qwen3-4B surface Lean warm start proves 84 problems, 82 of them untouched; the Qwen3-4B term warm start proves 29 problems, 28 of them untouched.

Templates on miniF2F. The same templates prove 56 of the 488 miniF2F problems, including every problem solved by the Qwen3-4B term warm start in eight samples. The Qwen3-4B surface Lean warm start proves a further 8.61% of the benchmark beyond the templates in eight samples, with a 95% interval of [6.15, 11.27].

Accepted proof forms. The successful term samples contain 35 distinct proofs. Of these, 13 use reflexivity and 22 apply `of_decide_eq_true`; both forms rely on kernel computation.

Automation. For the Qwen3-4B surface Lean warm start, 160 of 174 accepted samples call automation.

I Transfer from pretrained models

This section gives the from-scratch and random-model controls behind Section 4.3.

Qwen3-0.6B trained from scratch. Of the three warm starts, only the Qwen3-0.6B model was also trained from random initialisation. The proof loss of Qwen3-0.6B trained from scratch is 1.070 times that of the 390M grid model with the lowest proof loss (272.5 nats per theorem). The interval from resampling held-out modules is [1.056, 1.086] at the 95% level. With one run on each side, the contribution of run-to-run variation to this gap remains unresolved. The fitted Chinchilla form predicts a loss reduction of 1.4% when model size increases by a factor of 1.16. The proof loss of the Qwen3-0.6B warm start is lower than that of the 390M grid model by a factor of 3.44. The factor is close to the proof-loss ratio against Qwen3-0.6B trained from scratch (Section 4.3).

Random-network controls. To see whether the head start comes from the transferred embeddings or from the transformer body, we score randomly initialised models. Across all held-out theorems, the uniform predictor's proof loss is 70,389 nats per theorem, close to that of the randomly initialised Qwen3-0.6B model in Table 4. On a random subset of 3,000 held-out theorems, a random network with the Qwen3-0.6B architecture gives 73,900 nats per theorem, compared with 74,594 for a uniform predictor on those same theorems. Transferring only the Qwen3-0.6B embeddings gives 75,962, whereas the full warm start gives 13,204 before any training on terms, so the pretrained transformer body carries the head start.

J SCALING WITH MODEL SIZE

This section gives two views of the training grid behind the scaling comparisons in Section 4.4: loss against model size and the placement of models in parameter–token space.

Reading the model-size curves. A curve's slope approaches $-\alpha$ where the model-size term dominates (Fig. J.1).

Tokens per parameter. Each range gives tokens processed per total parameter across one language's grid; their overlap defines the shared tokens-per-parameter band below.

- Kernel terms: 0.54–2,445.5.
- Surface Lean: 0.07–97.9.
- English: 0.57–330.2.

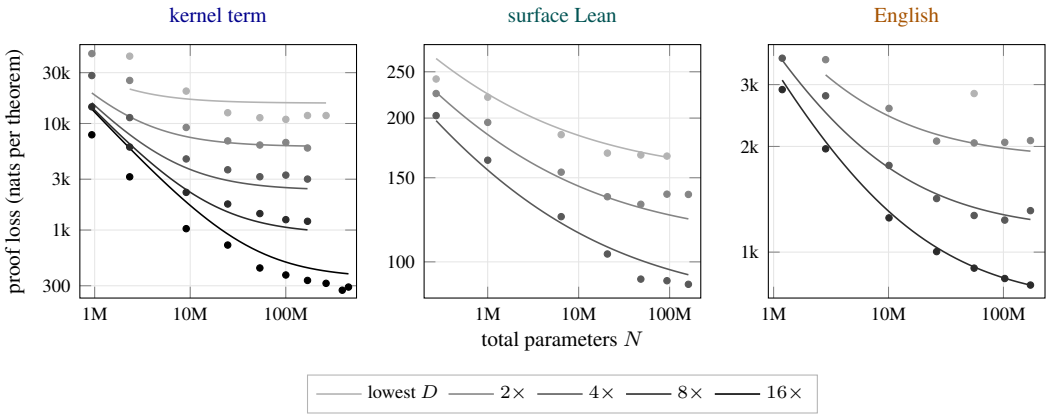

Figure J.1: Proof loss against total parameters, one curve per token budget (the legend gives each budget as a multiple of the lowest), one panel per language. Markers are models and lines are the fitted Chinchilla form. The rightmost term marker is Qwen3-0.6B trained from scratch (Appendix I).

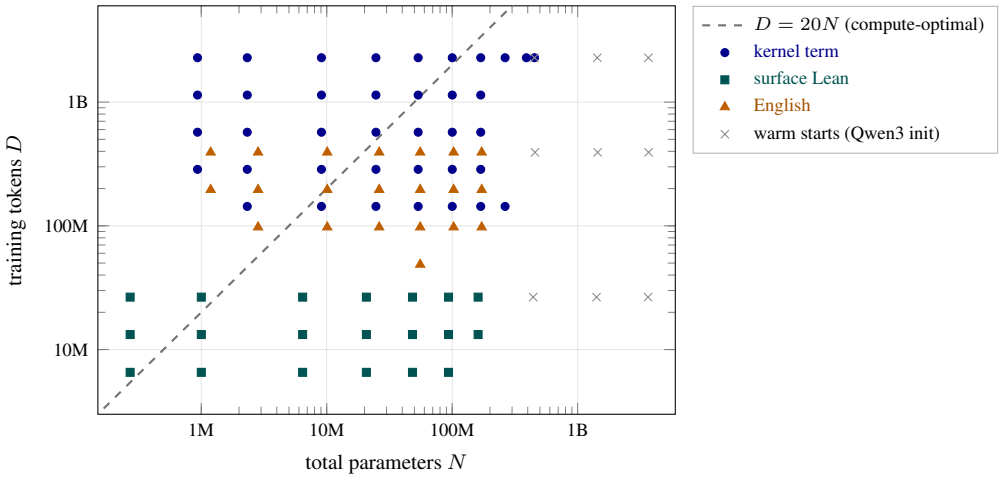

Figure J.2: Every from-scratch model by its parameter count and tokens processed, against the $D = 20N$ reference ray. Crosses are the Qwen3 warm starts, which use a different initialisation and enter no fit.

K Scaling fits and sensitivity checks

This section gives the fits, sensitivity checks and diagnostics supporting Section 4.4. The learning-rate probe shows that the grid's single learning rate inflates the term model-size exponent α over the probed widths (Table E.2). The relative fit error is the root mean square of log ratios of measured to fitted loss, reported as a percentage.

K.1 Fitting procedure

We fit Eq. (5) separately for each language by maximum a posteriori estimation in log space. Log-loss residuals are Gaussian with a fitted scale, and the exponents have the Gaussian priors in Section 3.4. We use zero-centred Gaussian priors with standard deviation 8 on $\log A$ and $\log B$, and 3 on the log residual scale. Posterior summaries are medians and central 90% intervals from 50,000 Laplace-approximation draws at the mode. The prior-free fit drops the two exponent priors and keeps everything else.

Sources of the exponent priors. For the two Lean languages, the prior centres and standard deviations are the medians and spread of the published code estimates (Table K.3). These estimates concern general programming languages, the nearest published proxy for Lean. For English, we centre the priors on Hoffmann et al. (2022)'s fit and use the same standard deviations.

K.2 Observed trends and limits

Term loss steepens with data. With $E \geq 0$, the Chinchilla form predicts that log proof loss flattens in log D at fixed model size (Eq. (5)). At a model size of 53.6M parameters, the measured term loss instead steepens: its local negative slope rises from 0.92 to 1.42 over the nominal token budgets (a concavity z-score of 5.63). A one-sided concavity test, with log-loss noise 0.05, rejects flattening at 4 of 6 term sizes, 1 of 6 surface Lean sizes and 0 of 6 English sizes.

Empirical slopes preserve the ordering. Table K.2 checks the observed rates of improvement without fitting the Chinchilla form. Across widths, the share of module resamples in which the shallowest term data slope is steeper than the steepest surface Lean slope is 100.0% (Table K.2). The share of module resamples in which the shallowest term data slope is steeper than the steepest English slope is 100.0%. The empirical model-size slopes have the same ordering at each token budget.

The floor checks preserve the ordering. With each language's floor held at its upper limit, the term intervals for both exponents stay above those of surface Lean and English, with or without exponent priors (Table K.3). With central 90% intervals, the smallest gap lies between the lower end of term's α interval and the upper end of English's, and is +0.068.

As each language's floor moves from zero to its upper limit, refitting with the exponent priors gives the following ranges of posterior medians:

- Kernel terms: α 0.965–0.967; β 1.366–1.388.
- Surface Lean: α 0.339–0.357; β 0.442–0.572.
- English: α 0.649–0.655; β 0.684–0.732.

The fitted data contribution usually dominates. At each model's N and D, the fitted data contribution $BD^{-\beta}$ exceeds the model-size contribution $AN^{-\alpha}$ in 28 of the 38 term models (Eq. (5)). At the term model with the lowest proof loss (390M parameters), the model-size contribution accounts for 10.0% of fitted loss.

K.3 Sensitivity conventions

The refits hold $E = 0$ and vary the following choices:

- **Full grid.** The fits of Section 4.4, with embeddings counted in N, with and without the exponent priors.
- **Non-embedding N.** Exclude embeddings from the parameter count to test sensitivity to vocabulary size.
- **Shared widths.** Keep only the seven widths all three languages share, reducing term models from 38 to 34.
- **Shared tokens-per-parameter band.** Keep models within 0.6–97.9 tokens per parameter, the range common to the three grids (Fig. J.2).
- **Shared epoch window.** Keep models within 0.49–1.62 epochs, since the original grids cover different epoch and update ranges (Table E.4).
- **Without repeated proofs.** Keep every model, but remove held-out proofs that exactly match a training proof before computing mean loss. For reference, the shares of held-out statements that match a training statement exactly are 0.34% for terms, 1.05% for surface Lean and 0.02% for English.
- **Nats per token.** The full-grid fits with loss in nats per token (Appendix L).
- **Wider amplitude priors.** Increase the log-amplitude prior standard deviation from 8 to 80. No exponent changes by more than 0.0539.

What the restricted grids identify. Term's α exceeds surface Lean's under every check (Table K.1). With the exponent priors, the comparison between the term and English model-size exponents remains unresolved in the shared tokens-per-parameter band and the shared epoch window. In the shared tokens-per-parameter band, the term estimate is 0.734 [0.261, 1.210] with the exponent priors and 1.492 without them. With the priors, the term interval contains the English estimate of 0.840.

The shared epoch window leaves each language at most two token budgets, so the β it reports is not an identified data-size exponent. Without the exponent priors, the term and English β intervals in the shared epoch window span zero. Term's β exceeds the other two languages' under every convention that identifies it.

Reading the empirical slopes. We fit least-squares lines to log proof loss against $\log D$ at fixed width and against $\log N$ at fixed nominal token budget. The slopes' uncertainty comes from 500 resamples of the 755 held-out modules, shared across the three languages. Under the Chinchilla form, a local negative slope is the exponent times that term's share of the loss (Eq. (5)).

Table K.1: Exponents under each sensitivity check. Entries are posterior medians with $E = 0$ and the exponent priors unless the row says otherwise. P, where computed, is the percentage of pairs of independent posterior draws in which term's α is the larger.

	α			P (%)		β		
check	term	surface	English	vs surface	vs English	term	surface	English
full grid	0.965	0.339	0.649	100.0	99.8	1.366	0.442	0.684
no exponent priors	1.121	0.334	0.669	100.0	99.9	1.339	0.441	0.679
non-embedding N	0.739	0.272	0.369	100.0	100.0	1.359	0.461	0.736
seven shared widths	0.975	0.339	0.649	100.0	99.7	1.295	0.442	0.684
tokens-per-parameter band	0.734	0.236	0.840	95.7	36.0	1.392	0.494	0.648
epoch window	0.519	0.360	0.580	96.9	26.7	1.232	0.297	-0.081
without repeated proofs	0.965	0.337	0.649	—	—	1.366	0.437	0.684
nats per token	0.979	0.340	0.651	—	—	1.378	0.442	0.685

Table K.2: Empirical rates of loss reduction (negative log-log slopes). Budget k is each language's k-th smallest nominal token budget. A dash marks a width or budget with fewer than three models.

	kernel term	surface Lean	English
slope in log D *at fixed width, by* d_{model}			
64	0.858	0.126	—
128	0.962	0.217	0.419
256	1.063	0.282	0.515
384	1.024	0.347	0.520
512	1.153	0.427	0.562
640	1.214	0.429	0.641
768	1.260	—	0.680
median over widths	1.063	0.315	0.541
slope in log N *at fixed nominal token budget*			
budget 1	0.262	0.069	0.129
budget 2	0.405	0.082	0.218
budget 3	0.532	0.133	0.255
median over token budgets	0.405	0.082	0.218

K.4 Floor limits and published exponents

Table K.3: Floor limits and published exponents. Our E values are one-sided 95% upper limits, and published floors are fitted values. The share is the upper limit as a percentage of the language's lowest measured proof loss.

	E, upper limit		
language	nats per theorem	nats per token	share (%)
kernel term	46.7	0.00657	17.1
surface Lean	28.3	0.87841	31.6
informalised English	91.6	0.19062	11.4

Published Chinchilla-form fits, 9 in all	α	β	E, nats per token
English (Hoffmann et al., 2022)	0.34	0.28	1.69
code (Luo et al., 2026)	0.485	0.298	0.220
7 languages (Yang et al., 2026)	0.221–0.845	1.129–1.350	—

L THE SAME LAWS IN NATS PER TOKEN

This section gives the exact conversion between loss units and checks it by refitting in nats per token, supporting the exponent comparison in Section 4.4.

The conversion leaves exponents unchanged. On a fixed held-out set, the two conventions differ by one constant per language: the mean proof tokens per held-out theorem T_c. Writing L_c for Eq. (5) in nats per theorem and L_c^{tok} for the same quantity per token gives

$$L_c(N, D) \;=\; T_c \cdot L_c^{\mathrm{tok}}(N, D). \tag{8}$$

The per-token rate is total nats divided by total proof tokens, so multiplying it by mean proof tokens per theorem gives mean nats per theorem exactly. Converting a per-token law to nats per theorem multiplies E, A and B by T_c, leaving α and β unchanged.

Refitting checks the unit conversion. We check the unit conversion by refitting all 38 term, 20 surface and 21 English models in nats per token (Table K.1). No exponent changes by more than 0.0247 under either prior setting. Each estimate lies within the corresponding interval from the other unit convention, preserving the full-grid comparisons (Section 4.4).