# SHELLELASTIC: SCALABLE ELASTIC PRETRAINING WITH SIMULTANEOUS MAXIMAL UPDATES AND HYPERPARAMETER TRANSFER

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Foundation models need to support diverse inference budgets, yet training a separate model for each budget duplicates pretraining effort. Elastic models train several nested sub-networks inside one weight-shared supernet, but existing recipes face three obstacles: variable per-step workloads that hinder distributed scaling, limited hyperparameter (HP) transfer across model sizes, and no guarantee that all sub-networks are maximally updated at once. We present SHELLELASTIC, an elastic transformer that organizes hidden dimensions into a dense core and successive shells, so that every hidden matrix is block lower triangular. One full-width forward pass contains every sub-network's exact hidden states as prefixes, so joint training keeps a fixed computation graph and costs about as much per step as training the widest sub-network alone. We further introduce $\mu$-ELASTIC, an extension of maximal update parameterization ($\mu$P) to SHELLELASTIC and other elastic architectures. Its scaling rules provably keep all sub-networks of a supernet maximally updated simultaneously ($\mu$-Internal), and keep doing so when the supernet is widened or given more shells; sweeps confirm that HPs tuned on a small non-elastic proxy stay near-optimal for elastic families and larger supernets ($\mu$-Transfer). A single SHELLELASTIC run produces sub-networks that outperform MatFormer and fully elastic baselines at matched parameter counts, and approach separately trained models that together take several times the steps.

## 1 INTRODUCTION

Foundation models need to support different compute budgets for different tasks and deployment platforms. Elastic training answers this need by optimizing several nested sub-networks inside one shared parameter set: slimmable networks and Once-for-All established the idea for vision models (Yu et al., 2018; Cai et al., 2020), MatFormer nests the FFN width of a transformer and underlies released on-device models (Devvrit et al., 2024; Gemma Team, 2025), and Flextron and Nemotron Elastic make pretrained LLMs elastic by post-training (Cai et al., 2024; Taghibakhshi et al., 2025).

Three problems remain when the elastic training recipe is carried into foundation-model pretraining.

*Scalability.* Standard recipes sample one or a few sub-networks per step (Devvrit et al., 2024; Yu & Huang, 2019), so the computation graph, the parameter access pattern and the workload change from step to step; under parameter sharding this variability complicates activation memory, recomputation and distributed scheduling, the very machinery that large-scale pretraining depends on. Training all sub-networks at every step fixes the workload but needs a forward and backward pass per sub-network, so the cost grows with the family size and erodes the saving that motivates elastic training.

*Transferability.* Hyperparameters (HPs) tuned at one model size need not remain effective at another, and repeating the search over learning rate, initialization scale and weight decay at every target scale cancels the compute that elastic training saves. Maximal update parameterization transfers HPs across widths for independently trained models (Yang et al., 2021); elastic models add two questions: can HPs tuned on a non-elastic proxy serve an entire elastic family, and can those tuned on a smaller elastic family serve a larger one? We refer to these two forms of transfer as $\mu$-Transfer for elastic models and seek one scaling rule that delivers both under parameter sharing.



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.

*Simultaneous maximal updates.* Existing elastic training recipes do not generally ensure maximal updates for all sub-networks simultaneously.[1] Without width-dependent scaling, HPs that yield well-scaled feature updates in one sub-network make updates vanish or blow up in another, and re-tuning one global value only trades one width against another, so some widths stay mis-scaled. The requirement is therefore a shared parameterization that keeps every width maximally updated simultaneously despite parameter sharing; we call this property $\mu$-Internal for elastic models.

We address the first problem with a new architecture, SHELLELASTIC, and the other two with $\mu$-ELASTIC. SHELLELASTIC (Figure 1) partitions the residual stream into a dense core followed by ordered shells, each a multiple of a base width $g$, and each supported sub-network is the core plus a prefix of the shells. Every hidden matrix is block lower triangular: a shell reads the core, the earlier shells and itself, but never a later shell. Trunk normalization acts within each group, attention heads align with the group boundaries, and every supported width has its own readout. One full-width forward pass therefore contains the exact hidden states of every sub-network as prefixes (Proposition 1), and the per-step compute stays close to that of a single full-width pass however many sub-networks are trained, so every step has a fixed workload, as in non-elastic training.

$\mu$-ELASTIC extends maximal update parameterization ($\mu$P) (Yang et al., 2021) to elastic models with two factors: a $\mu$-Transfer factor $m = g/d_0$ that carries HPs from a non-elastic proxy of width $d_0$ to base width $g$, and a $\mu$-Internal factor for the different widths inside one shared model. Every group reads the same input coordinates in every sub-network that contains it, so we set its initialization, learning rate and weight decay from this fan-in, while each readout uses its own input width (Table 1). This keeps every group maximally updated in all sub-networks at once (Theorem 1), which uniform scaling across groups cannot do in general (Proposition 2). The same two factors apply to MatFormer and to fully elastic models (Appendix E).

Our main contributions are summarized as follows:

- ★ **Scalable elastic architecture.** We propose SHELLELASTIC, an elastic transformer whose single full-width forward pass yields the exact hidden states of every sub-network as prefixes, so every step has a fixed workload rather than a varying one.
- ★ **$\mu$-ELASTIC parameterization.** We introduce $\mu$-ELASTIC, which transfers HPs from a non-elastic proxy to an elastic model and from smaller to larger elastic models ($\mu$-Transfer), and keeps all sub-networks of a supernet maximally updated at once ($\mu$-Internal).
- ★ **Theoretical and empirical validation.** We prove that $\mu$-ELASTIC keeps all sub-networks maximally updated at both levels of transfer; sweeps confirm that transferred HPs stay near-optimal and that $\mu$-ELASTIC beats the standard parameterization on every elastic family.
- ★ **Comprehensive ablation studies.** We search architectures at equal parameters and ablate training schedules, loss weights, normalization scope, readout design, in-place distillation, sub-network ensembles and stage-residual gradient scaling to guide elastic pretraining.

## 2 RELATED WORK

**Elastic models.** Elastic models train nested sub-networks in one shared parameter set, so one run serves many inference budgets (Yu et al., 2018; Cai et al., 2020). MatFormer (Devvrit et al., 2024) nests only the FFN hidden width of a transformer and needs one forward pass per width. The concurrent StairFormer (Trost et al., 2026) nests each sub-network as a prefix of one forward pass and instantiates the nesting with equal-width blocks; SHELLELASTIC keeps this structure but chooses the supported widths and normalizes within groups, not over cumulative prefixes. None of these works derives HP scaling rules across nested widths or keeps all sub-network maximally updated.

**$\mu$-parameterization.** Under the standard parameterization (Sohl-Dickstein et al., 2020), optimal HPs drift with width; $\mu$P (Yang et al., 2021) gives every layer maximal $\Theta(1)$ feature updates as width grows, so HPs tuned on a small proxy transfer zero-shot; CompleteP (Dey et al., 2026) and Mlodozeniec et al. (2026) extend this transfer to depth, batch size and token budget. These rules assume separately trained models with one fan-in per tensor; $\mu$-ELASTIC extends them to the shared tensors of an elastic supernet. Appendix A gives a comprehensive review.

[1] A parameter group is *maximally updated* when the pre-activations it produces are $\Theta(1)$ at initialization and move by $\Theta(\eta_0)$ after one optimizer step, with $\eta_0$ the base learning rate before width scaling, so that its contribution neither vanishes nor blows up as the width grows (Yang & Hu, 2021; Yang et al., 2021).

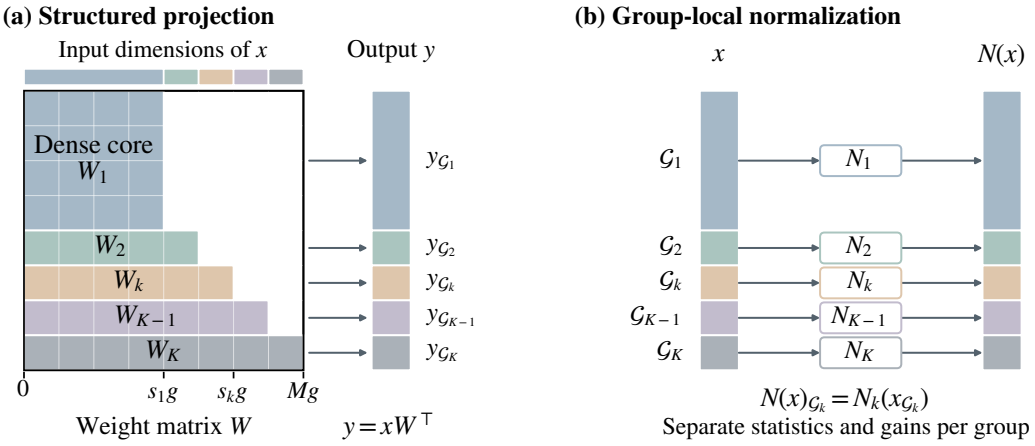


Figure 1: **Core and shells preserve supported prefixes.** (a) Each colored block row produces output group $\mathcal{G}_k$ from the input prefix ending at $s_k g$. Group $\mathcal{G}_1$ is the dense core; subsequent groups are shells. (b) $N_k$ normalizes group $\mathcal{G}_k$ using its own statistics.

## 3 METHOD

In the SHELLELASTIC architecture, all sub-networks reuse hidden-state prefixes from one full-width trunk pass; $\mu$-ELASTIC, a parameterization based on each weight's active input width, enables reuse of base optimization settings across model sizes.

### 3.1 THE SHELLELASTIC ARCHITECTURE

**Core, shells, and supported widths.** We organize hidden dimensions into a dense ***core*** followed by successive ***shells*** (Figure 1). For base width $g$, choose supported sub-network width indices

$$\mathcal{S} = \{s_1 < \cdots < s_K = M\} \subseteq \{1, \ldots, M\}, \qquad D^{(s)}_{\text{model}} = sg, \quad D_{\text{model}} = Mg. \tag{1}$$

With $s_0 = 0$, these boundaries define contiguous groups $\mathcal{G}_k = \{s_{k-1}g + 1, \ldots, s_k g\}$ of size $(s_k - s_{k-1})g$. The core is $\mathcal{G}_1$, and the shells are $\mathcal{G}_2, \ldots, \mathcal{G}_K$. Sub-network $s_k$ contains the first $k$ groups.

**Projections and normalization.** Write a token's hidden state as a row vector $x$, with $x_{:d}$ denoting its first $d$ dimensions. A square projection is $y = xW^\top$, where $W \in \mathbb{R}^{D_{\text{model}} \times D_{\text{model}}}$. For output group $k$, the block row $W_k \in \mathbb{R}^{|\mathcal{G}_k| \times s_k g}$ reads only the first $s_k g$ input dimensions:

$$y_{\mathcal{G}_k} = x_{:s_k g} W_k^\top. \tag{2}$$

This makes $W$ block lower triangular at the group boundaries (Figure 1a).

Group-local normalization uses a separate operator $N_k$ and statistics within each group (Figure 1b). For RMSNorm (Zhang & Sennrich, 2019):

$$\left[N_k(x_{\mathcal{G}_k})\right]_j = a_j \frac{x_j}{\sqrt{|\mathcal{G}_k|^{-1} \sum_{u \in \mathcal{G}_k} x_u^2 + \epsilon_N}}, \qquad j \in \mathcal{G}_k. \tag{3}$$

Here $a_j$ is a learned gain and $\epsilon_N > 0$ is a numerical stabilizer. The combined normalization satisfies $N(x)_{\mathcal{G}_k} = N_k(x_{\mathcal{G}_k})$. Both operations preserve prefixes ending at the boundaries in $\mathcal{S}$. Cuts within a group generally change the retained outputs, so sub-networks can be extracted only at these boundaries. Thus, $\mathcal{S}$ determines the block-triangular pattern of every projection, the normalization groups, and the available sub-network widths.

**Attention and FFN.** Both branches (Figure 5 in Appendix B) use the group-local normalization of Equation 3 as their pre-norm. Attention is elastic in its number of heads: sub-network $s$ uses the first $D^{(s)}_{\text{model}}/d_{\text{head}} = sg/d_{\text{head}}$ heads, keeping the head dimension $d_{\text{head}}$ fixed. We require $g$ to be a multiple of $d_{\text{head}}$ so that supported widths retain complete heads. The query, key, value, gate, and output projections follow Equation 2; they constrain dependencies among hidden dimensions before

Table 1: $\mu$-ELASTIC **for** SHELLELASTIC**.** Scaling rules set by the $\mu$-Transfer factor $m$, the $\mu$-Internal factor $s$ (Equation 7), and the FFN expansion $e = D_{\text{ffn}}/D_{\text{model}}$; $s$ is the output group's supported boundary for hidden weights and the sub-network width index for readouts. Initialization gives nominal weight standard deviations, or initial values for gains and scalars.

| Parameter | Active input width | Initialization | Learning rate | Weight decay | Multiplier |
|---|---|---|---|---|---|
| Token embedding $E$ | $V$ (fixed) | $\sigma_0$ | $\eta_0$ | $\lambda_0$ | 1 |
| Attention; FFN up/gate | $sg$ | $\sigma_0/\sqrt{ms}$ | $\eta_0/(ms)$ | $\lambda_0 ms$ | 1 |
| FFN down | $esg$ | $\sigma_0\sqrt{e}/\sqrt{ms}$ | $\eta_0/(ms)$ | $\lambda_0 ms$ | $1/e$ |
| Readout $U_s$ | $sg$ | $\sigma_0/\sqrt{ms}$ | $\eta_0/(ms)$ | $\lambda_0 ms$ | $\alpha_s$ |
| Norm gains; readout scalar $\alpha_s$ | 1 (elementwise) | 1 | $\eta_0$ | 0 | 1 |

and after attention. Query/key normalization and rotary position transformations (Su et al., 2024) act within each head. Each active head $a$ performs causal self-attention:

$$\text{Attn}(x) = \left[\text{concat}_a\left(\text{softmax}\left(Q_a K_a^\top/\sqrt{d_{\text{head}}} + C\right) V_a\right) \odot \text{sigmoid}(xW_G^\top)\right] W_O^\top, \quad (4)$$

where $C$ is the causal mask, $Q_a, K_a, V_a$ come from the corresponding projections of $x$ and concatenation runs over the active heads. Adding heads leaves earlier heads' attention outputs unchanged; the structured output projection then preserves the supported hidden-state prefixes.

We use a SwiGLU FFN (Shazeer, 2020) with expansion $e = D_{\text{ffn}}/D_{\text{model}}$ and integer base FFN width $eg$, so sub-network $s$ has $D_{\text{ffn}}^{(s)} = eD_{\text{model}}^{(s)}$. All three maps use the same group dependency pattern. The up and gate weights have shape $D_{\text{ffn}} \times D_{\text{model}}$, with group boundaries $s_k g$ on the input axis and $es_k g$ on the output axis. The down weight has shape $D_{\text{model}} \times D_{\text{ffn}}$. The FFN computes

$$\text{FFN}(x) = \left[\text{SiLU}(xW_1^\top) \odot (xW_3^\top)\right] W_2^\top. \quad (5)$$

Elementwise operations and residual additions preserve the same prefixes.

**Embedding and readout.** Sub-network $s$ uses $sg$-dimensional token embeddings and computes its logits from the first $sg$ dimensions of the $L$-layer trunk output $x^{(L)}$:

$$z_s = \alpha_s \, N_s^{\text{out}}(x_{:sg}^{(L)}) U_s^\top, \qquad s \in \mathcal{S}. \quad (6)$$

Here $\alpha_s$ is a learned logit scale, $N_s^{\text{out}}$ normalizes over all $sg$ selected dimensions, and $U_s \in \mathbb{R}^{V \times sg}$ is the readout matrix for a vocabulary of size $V$. Figure 4 in Appendix B shows the complete model.

**Proposition 1** (Exact reuse at supported widths)**.** *For fixed model parameters, let $E_s$ be the first $sg$ columns of the full-width embedding table, $T_s$ the $L$-layer trunk restricted to width $sg$, and $P_s$ the operator selecting the first $sg$ hidden dimensions. For any token sequence $u$, evaluating sub-network $s$ directly gives the same hidden states as extracting its prefix from sub-network $t$:*

$$T_s(E_s[u]) = P_s(T_t(E_t[u])), \qquad s, t \in \mathcal{S}, \quad s \le t.$$

Applying sub-network $s$'s normalization and readout (Equation 6) to either result gives identical logits. Thus, one full-width trunk evaluation yields the same joint loss and parameter gradients as separate evaluations of all supported sub-networks. Appendix C proves the result by composing the prefix-preserving trunk operations across the $L$ layers.

### 3.2 $\mu$-ELASTIC: PARAMETERIZING ELASTIC MODELS

We introduce $\mu$-ELASTIC to address the two optimization requirements: transferring HPs from a non-elastic proxy to an elastic family and between elastic model sizes ($\mu$-Transfer), and maintaining maximal updates across sub-networks within each shared model ($\mu$-Internal).

**Composing transfer and internal scaling.** Let a non-elastic proxy of width $d_0$ supply the base initialization scale $\sigma_0$, base learning rate $\eta_0$, and weight decay $\lambda_0$. For base width $g$, define

$$m = \frac{g}{d_0}, \qquad q(s) = \underbrace{m}_{\mu\text{-Transfer}} \cdot \underbrace{s}_{\mu\text{-Internal}} = \frac{sg}{d_0}, \quad s \in \mathcal{S}. \quad (7)$$

For a hidden weight, $s$ is the supported boundary of its output group, whose input width stays $sg$ ($esg$ for an FFN down projection) in every sub-network that contains it, so one rule based on $q(s)$ serves all of them. The readout of sub-network $s$ uses the same ratio.

**Parameterization and optimization rules.** Table 1 summarizes initialization, learning rate, weight decay, and forward multipliers by active input width for AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2019), following the per-layer scaling of Everett et al. (2024). Width is the only elastic axis, since all sub-networks share the depth $L$, minibatches and number of training steps; these shared quantities follow existing scaling rules for depth (Dey et al., 2026), batch size and token budget (Mlodozeniec et al., 2026). The learning rate and weight decay scalings cancel, so AdamW applies the same relative decay to every decayed parameter group. The embedding is a lookup over a fixed vocabulary of size $V$, so it uses the base $(\sigma_0, \eta_0, \lambda_0)$. For FFN down projections, the $\sqrt{e}$ initialization factor and $1/e$ forward multiplier account for the expanded input width $esg$. The learned scalars $\alpha_s$ in Equation 6 adjust logit coefficients beyond width scaling. Norm gains and $\alpha_s$ are initialized to one and use learning rate $\eta_0$ with zero weight decay. The *standard parameterization* (SP) sets $m = s = e = 1$ only in the scaling rules and fixes $\alpha_s = 1$, preserving the architecture.

**Theoretical guarantees.** The statements below are proved in Appendix D under the standard heuristic assumptions of $\mu$P (Yang et al., 2021): before the scaling of Table 1 is applied, the pre-activations of a group with active input width $sg$ are $\Theta\left(\sqrt{sg}\right)$ at initialization and move by $\Theta(sg)$ after one step, which is why the initialization and the learning rate carry different exponents.

**Theorem 1** ($\mu$-Internal: simultaneous maximal updates)**.** *Under the assumptions of Appendix D and the rule of Table 1, consider a hidden group with supported boundary $s$ and any sub-network $t \in \mathcal{S}$ with $t \geq s$. In every layer, its pre-activations inside sub-network $t$ (i) have $\Theta(1)$ coordinates at initialization and (ii) move by $\Theta(\eta_0)$ after one AdamW step on that group, with constants independent of $s$, $t$, $M$, and $m$. The logits of every readout $U_s$ have the same two properties.*

Equation 7 is what makes this possible: a group with boundary $s$ sums over exactly $sg$ input coordinates in every sub-network containing it, so one initialization scale and one learning rate satisfy both conditions for all $t \geq s$ at once. Training several sub-networks jointly does not change this, since they all feed a group the same input and Adam is invariant to the gradient scale.

**Proposition 2** (A shared rule leaves groups mismatched)**.** *If all groups of a hidden matrix share one initialization variance, learning rate and multiplier, the groups with boundaries $s < s'$ differ by factors $\Theta\left(\sqrt{s'/s}\right)$ in pre-activation size and $\Theta(s'/s)$ in update size, whatever the shared values. For unbounded $M/s_1$, no shared rule thus keeps every group maximal in the sense of Theorem 1.*

Under SP, the groups of a hidden matrix share $(\sigma_0, \eta_0)$, so by Proposition 2 re-tuning only trades the core against the outermost group, which stay $\sqrt{M/s_1}$ and $M/s_1$ apart in initialization and update; the best HPs are compromises depending on $M$ and $\mathcal{S}$. Figure 2 and Table 3 show both effects: the SP optimum departs from the proxy's, and SP trails $\mu$-ELASTIC even after re-tuning.

**Corollary 1** ($\mu$-Transfer: two-level transfer)**.** *Theorem 1 holds for every base width $g = md_0$ and every supported set $\mathcal{S} \subseteq \{1, \ldots, M\}$. Hence the values $(\sigma_0, \eta_0, \lambda_0)$ tuned on the width-$d_0$ proxy (i) keep every group and readout of a* SHELLELASTIC *supernet maximally updated, and (ii) keep doing so when the supernet is widened (larger $m$) or given more shells (larger $M$).*

As with $\mu$P, the corollary fixes the scales of the initialization and the updates, not the optimal HP values; Figure 2 verifies that the proxy's tuned HP values remain near-optimal at both levels.

**$\mu$-ELASTIC for other elastic architectures.** $\mu$-ELASTIC carries over to other elastic architectures, but $m$ and $s$ are defined per architecture. MatFormer (Devvrit et al., 2024) keeps the model width fixed and nests only the FFN hidden width, so only its FFN down projection has a fan-in that varies with the sub-network. It uses $m = D_{\text{model}}/d_0$, with $s$ the sub-network's FFN hidden width in units of a base FFN width; the down projection takes the factor $ms$, and the other hidden matrices and the readout take $m$ (Table 5). Fully elastic models, a transformer adaptation of slimmable networks (Yu et al., 2018), slice shared dense matrices along the residual, attention and FFN widths, so every hidden matrix has a varying fan-in; their $m$ and $s$ follow Equation 7, and every hidden matrix takes the factor $ms$ (Table 4). StairFormer (Trost et al., 2026) with equal-width blocks uses Table 1 with $\mathcal{S} = \{1, \ldots, M\}$ for its hidden matrices. Theorem 1 and Corollary 1 extend to all three (Appendix D.5), and Figure 6 compares MatFormer, fully elastic models and SHELLELASTIC.

### 3.3 TRAINING AND INFERENCE

**Recipe.** (1) Tune $(\sigma_0, \eta_0, \lambda_0)$ on a non-elastic proxy of width $d_0$. (2) Choose the base width $g$ and the supported set $\mathcal{S}$. (3) Compute $m = g/d_0$ and instantiate the parameterization in Table 1. (4) Train the supernet on the joint objective below.

**Joint objective.** At step $\tau$, with a selected set of widths $\mathcal{A}_\tau \subseteq \mathcal{S}$ and non-negative loss weights $\omega_s$,

$$\mathcal{L}_\tau = \frac{\sum_{s\in\mathcal{A}_\tau} \omega_s \, \ell_s}{\sum_{s\in\mathcal{A}_\tau} \omega_s}, \qquad \ell_s = \mathrm{CE}(z_s, y) + \lambda_z \, \mathbb{E}\left[\left(\log \textstyle\sum_v e^{z_{s,v}}\right)^2\right], \tag{8}$$

where $z_s$ are the logits of Equation 6, $\lambda_z$ is the $z$-loss coefficient, the expectation runs over the non-padded target positions, and every $\ell_s$ backpropagates into the shared trunk. The narrower widths may be selected by any rule, but every $\mathcal{A}_\tau$ contains the full width $M$, so the trunk always runs at width $D_{\mathrm{model}}$ and every trunk parameter receives a gradient at every step.

**Cost and scalability.** Let $C_{\mathrm{trunk}}(s)$ and $C_{\mathrm{out}}(s)$ be the forward costs of the trunk and of one terminal norm and readout at width $sg$. Training the family jointly and training each sub-network separately cost

$$C_{\mathrm{joint}} = C_{\mathrm{trunk}}(M) + \sum_{s\in\mathcal{S}} C_{\mathrm{out}}(s), \qquad C_{\mathrm{separate}} = \sum_{s\in\mathcal{S}} \Big[C_{\mathrm{trunk}}(s) + C_{\mathrm{out}}(s)\Big]. \tag{9}$$

By Proposition 1, one trunk term serves every width, whereas $C_{\mathrm{separate}}$ pays one per width, so the gap grows with $|\mathcal{S}|$ and $C_{\mathrm{joint}}$ stays comparable to training the widest sub-network alone. The same holds for memory: all readout gradients combine in one trunk backward pass, so the trunk's activations, parameters and optimizer state are those of the widest one. Training all sub-networks at every step works best (Appendix K) and gives every step the same computation graph, parameter access pattern and workload. In MatFormer and fully elastic models, hidden states are not nested, so each width needs its own trunk pass: sampling widths per step (Devvrit et al., 2024; Yu & Huang, 2019) varies the workload, while training all widths at every step (Yu et al., 2018) pays $C_{\mathrm{separate}}$.

**Inference.** For a chosen width $s \in \mathcal{S}$, we retain the embedding prefix, the core and shells up to $s$, and the matching terminal norm and readout, keeping the forward multipliers of Table 1. By Proposition 1, the extracted model reproduces its supernet logits exactly and runs as a standalone transformer of width $sg$ without further training.

## 4 EXPERIMENTS

After describing the setup (Section 4.1), we evaluate the sub-networks of one SHELLELASTIC run against parameter-matched MatFormer, fully elastic and separately trained models, and $\mu$-ELASTIC against SP on every elastic family (Section 4.2). We then compare SHELLELASTIC with StairFormer and run an architecture search at equal parameters and width count (Section 4.3); finally, we ablate the training and design choices to derive practical guidance for elastic pretraining (Section 4.4).

### 4.1 EXPERIMENTAL DETAILS

**Models.** Our primary model is a SHELLELASTIC supernet with base width $g = 128$ and supported set $\mathcal{S} = \{4, 5, 6, 7, 8\}$: a 512-dimensional core followed by four 128-dimensional shells, giving five sub-networks of widths 512 to 1024. We report main results at three scales, with sub-networks of about 119M–285M, 375M–906M and 675M–1.69B parameters (SHELLELASTIC widths 512–1024, 1024–2048 and 1024–2048, respectively). All models are 32-layer transformers with gated attention and SwiGLU FFNs. Tables 6 and 7 in Appendix F list all configurations.

**Datasets.** All models are trained on DCLM-Baseline (Li et al., 2024), a filtered Common Crawl corpus, with the GPT-NeoX-20B tokenizer (Black et al., 2022) and a sequence length of 2,048. By default, a run takes 25,000 steps of 128 sequences (6.55B tokens). This budget gives every sub-network more than 22 tokens per parameter, above the compute-optimal ratio of about 20 (Hoffmann et al., 2022). We monitor the training and validation losses (per-token cross-entropy in nats) of every sub-network, the latter on held-out C4 text (Raffel et al., 2020; Dodge et al., 2021). Following

Table 2: Final validation loss and mean 5-shot accuracy over 13 downstream tasks at three scales, labeled by sub-network size. Entries at the 119M–285M scale average three runs with the same seed; the others are single runs. †Five separately trained models, together taking 5× the optimizer steps of any other row. Bold marks the best of the four methods that train all five sizes in one run.

| | | Val. loss ↓ | | | | | | 5-shot acc. (%) ↑ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Method | Low | Medium | High | Extra | Max | Avg. | Low | Medium | High | Extra | Max | Avg. |
| 119M–285M | Standalone† | 3.369 | 3.308 | 3.264 | 3.219 | 3.191 | 3.270 | 35.6 | 35.8 | 36.6 | 37.1 | 37.1 | 36.5 |
| | Sampled standalone | 3.667 | 3.609 | 3.583 | 3.532 | 3.524 | 3.583 | 33.8 | 33.4 | 34.0 | 34.2 | 34.4 | 34.0 |
| | MatFormer | 3.425 | 3.357 | 3.325 | 3.312 | 3.314 | 3.347 | 35.1 | 35.2 | 35.8 | 35.9 | 35.9 | 35.6 |
| | Fully elastic | 3.429 | 3.354 | 3.315 | 3.295 | 3.290 | 3.336 | 34.9 | 35.5 | 35.7 | 35.9 | 36.0 | 35.6 |
| | SHELLELASTIC | **3.409** | **3.338** | **3.294** | **3.264** | **3.240** | **3.309** | **35.5** | **36.6** | **36.6** | **36.9** | **36.9** | **36.5** |
| 375M–906M | Standalone† | 3.006 | 2.964 | 2.925 | 2.899 | 2.869 | 2.932 | 40.3 | 41.4 | 42.3 | 42.8 | 43.1 | 42.0 |
| | Sampled standalone | 3.207 | 3.171 | 3.141 | 3.108 | 3.092 | 3.144 | 37.5 | 37.7 | 37.9 | 39.0 | 39.1 | 38.2 |
| | MatFormer | 3.074 | 3.007 | 2.977 | 2.965 | 2.965 | 2.997 | **40.8** | 40.5 | 40.6 | 41.5 | 41.9 | 41.1 |
| | Fully elastic | **3.053** | **2.996** | 2.968 | 2.953 | 2.949 | 2.984 | 40.0 | 41.0 | 42.1 | 42.1 | 42.0 | 41.4 |
| | SHELLELASTIC | **3.053** | 3.000 | **2.967** | **2.943** | **2.924** | **2.977** | **40.8** | **41.8** | **42.4** | **42.7** | **43.2** | **42.2** |
| 675M–1.69B | Standalone† | 2.912 | 2.859 | 2.821 | 2.793 | 2.767 | 2.830 | 43.4 | 43.9 | 44.7 | 46.2 | 46.1 | 44.8 |
| | Sampled standalone | 3.037 | 2.998 | 2.970 | 2.956 | 2.924 | 2.977 | 40.2 | 40.8 | 41.6 | 41.0 | 43.2 | 41.4 |
| | MatFormer | **2.914** | 2.872 | 2.849 | 2.839 | 2.838 | 2.862 | 42.8 | 43.3 | 43.4 | 44.0 | 43.6 | 43.4 |
| | Fully elastic | 2.940 | 2.884 | 2.854 | 2.838 | 2.833 | 2.870 | 42.1 | 43.4 | **43.9** | 44.2 | **44.6** | 43.6 |
| | SHELLELASTIC | **2.914** | **2.866** | **2.835** | **2.813** | **2.796** | **2.845** | **43.7** | **43.7** | **43.9** | **44.3** | 44.5 | **44.0** |

OLMoE (Muennighoff et al., 2025) and DCLM, we also probe 5-shot accuracy on ARC-Easy (Clark et al., 2018), PIQA (Bisk et al., 2020) and HellaSwag (Zellers et al., 2019) during training and evaluate the final sub-networks on 13 datasets spanning world knowledge, symbolic problem solving, commonsense reasoning, language understanding and reading comprehension (Appendix G).

**Baselines.** We compare SHELLELASTIC against five baselines, each providing networks of five sizes: Low, Medium, High, Extra and Max. (i) *MatFormer* (Devvrit et al., 2024) nests only the FFN hidden width. (ii) *Fully elastic*, a transformer adaptation of slimmable networks (Yu et al., 2018), nests the residual, attention and FFN widths by slicing shared dense matrices. (iii) *StairFormer* (Trost et al., 2026) shares the prefix structure of SHELLELASTIC but uses equal-width groups and normalizes over cumulative prefixes. (iv) *Standalone* trains five separate models, each on the full budget, and thus takes 5× the steps of one elastic run. (v) *Sampled standalone*, the standalone baseline of MatFormer, trains the five *Standalone* models within the steps of one elastic run. MatFormer, the fully elastic model and sampled standalone sample one sub-network uniformly at random per step, whereas StairFormer and SHELLELASTIC train all sub-networks in one full-width pass at every step. **In the main results, all baselines match the SHELLELASTIC sub-networks in parameter count to within 3% (Table 9).** Appendix F details the baselines' architecture and training, and Appendix E gives their parameterization.

**Hyperparameters.** All models are trained with AdamW. Unless otherwise specified, they use the base learning rate $\eta_0 = 2 \times 10^{-3}$, initialization scale $\sigma_0 = 3.3 \times 10^{-3}$ and weight decay $\lambda_0 = 0.01$, with $\mu$-ELASTIC for the elastic models and $\mu$P for the non-elastic ones, both with $d_0 = 128$. SHELLELASTIC has $m = 1$ and trains all five widths at every step with loss weights $(1, 1, 1, 1, 4)$ (Equation 8). Main results at the 119M–285M scale are averaged over three runs; compute constraints limit the 375M–906M and 675M–1.69B scales to one run. Ablations vary only the setting under study; Table 8 in Appendix F lists the remaining HPs and the hardware.

### 4.2 MAIN RESULTS

**Comparison with baselines across scales.** Table 2 compares SHELLELASTIC with MatFormer, the fully elastic model and the two standalone baselines at three scales (curves and per-task results in Appendix H); all three elastic models use $\mu$-ELASTIC (Appendix E). SHELLELASTIC is the supernet of Section 4.1 at 119M–285M, doubles its base width to $g = 256$ at 375M–906M, and also raises the FFN expansion from $e = 1$ to 4 at 675M–1.69B; these two scales train for 50,000 and 100,000 steps of 256 sequences (Appendix F). Among the methods that train all five sizes in one run, SHELLELASTIC has the lowest mean validation loss (0.027, 0.007 and 0.017 nats below the

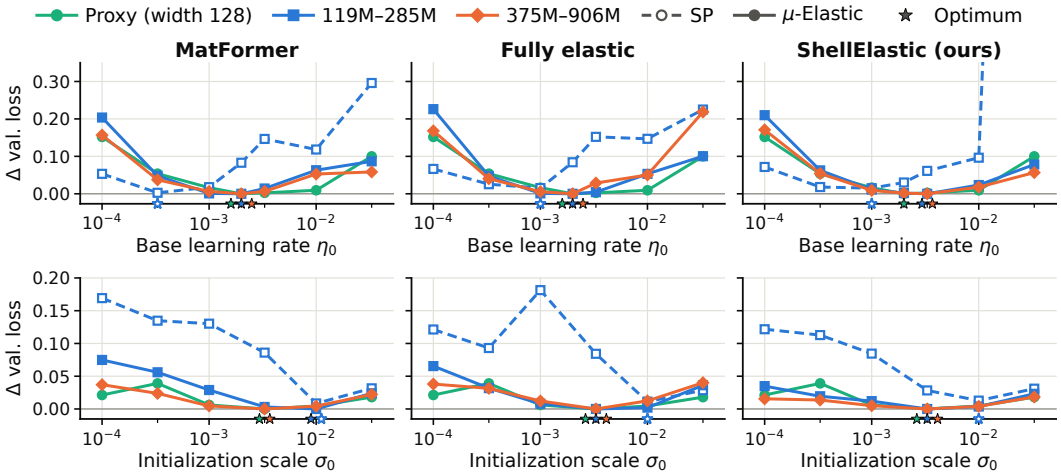


Figure 2: Sweeps of the learning rate $\eta_0$ and initialization scale $\sigma_0$, the other at its base value, for the proxy and each elastic model under $\mu$-ELASTIC and SP. $\Delta$ val. loss is the mean validation loss of the five sub-networks minus the minimum of the $\mu$-ELASTIC line of the same model and scale (for the proxy, its own line). Appendix I shows the sweeps for each sub-network.

Table 3: Effect of $\mu$-ELASTIC on each elastic model at the 119M–285M scale. For each model, the first row is the best of the 12 SP configurations of Figure 2 (lowest mean validation loss), and the w/ $\mu$-ELASTIC row is its result in Table 2, with the base values of Section 4.1 and no re-tuning.

| | Train loss ↓ | | | | | | Val. loss ↓ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Low | Medium | High | Extra | Max | Avg. | Low | Medium | High | Extra | Max | Avg. |
| MatFormer | 3.261 | 3.184 | 3.157 | **3.140** | **3.139** | 3.176 | 3.434 | 3.360 | 3.327 | 3.314 | **3.311** | 3.349 |
| w/ $\mu$-ELASTIC | **3.254** | **3.182** | **3.156** | 3.141 | 3.144 | **3.175** | **3.425** | **3.357** | **3.325** | **3.312** | 3.314 | **3.347** |
| Fully elastic | 3.272 | 3.192 | 3.154 | 3.130 | 3.126 | 3.175 | 3.445 | 3.368 | 3.325 | 3.303 | 3.297 | 3.348 |
| w/ $\mu$-ELASTIC | **3.257** | **3.178** | **3.145** | **3.123** | **3.120** | **3.165** | **3.429** | **3.354** | **3.315** | **3.295** | **3.290** | **3.336** |
| SHELLELASTIC | 3.247 | 3.180 | 3.136 | 3.104 | 3.079 | 3.149 | 3.423 | 3.354 | 3.308 | 3.275 | 3.249 | 3.322 |
| **w/ $\mu$-ELASTIC** | **3.230** | **3.164** | **3.123** | **3.094** | **3.071** | **3.137** | **3.409** | **3.338** | **3.294** | **3.264** | **3.240** | **3.309** |

best baseline) and the highest mean 5-shot accuracy at every scale, and is best or tied-best at 14 of the 15 tiers on each metric. Its lead grows with size, from at most 0.016 nats at Low to 0.025–0.050 at Max, likely because the Max-only parameters of the elastic baselines receive a gradient on only one step in five on average, whereas every parameter of SHELLELASTIC receives one at every step. At a fixed step budget, weight sharing also helps: the fully elastic model beats sampled standalone, its unshared counterpart, by 0.11 to 0.25 nats on average. With a fifth of their optimizer steps, SHELLELASTIC comes within 0.039, 0.045 and 0.015 nats of the standalone models in mean validation loss and within 0.8 points in mean 5-shot accuracy, matching or exceeding them at the two smaller scales. Appendix F.4 analyzes the training and inference cost of all methods at 119M–285M.

**Transferability of $\mu$-ELASTIC.** Under $\mu$-ELASTIC, the proxy's optimum, $\eta_0 = 2 \times 10^{-3}$ and $\sigma_0 = 3.3 \times 10^{-3}$ (the base values of Section 4.1), transfers to all three elastic models at both levels of $\mu$-Transfer (Figure 2, Corollary 1). At the 119M–285M scale, both values are optimal or adjacent to the optimum ($\eta_0$: exact for MatFormer and the fully elastic model, adjacent for SHELLELASTIC; $\sigma_0$: exact for the fully elastic model and SHELLELASTIC, adjacent for MatFormer). At the 375M–906M scale, which enlarges $m$, $M$ or both and doubles the batch size and steps (Appendix F), the same holds, and $\sigma_0$ becomes exact for MatFormer too. The base values are thus within 0.003 nats of the best point of every $\mu$-ELASTIC sweep and within 0.005 nats for every sub-network (Appendix I). Under SP, the optimum departs from the proxy's, as Section 3.2 anticipates: the optimal $\eta_0$ falls to $10^{-3}$ (the fully elastic model and SHELLELASTIC) or $3.3 \times 10^{-4}$ (MatFormer), the optimal $\sigma_0$ rises to $10^{-2}$, and at the base values SP is 0.08 nats worse for MatFormer and the fully elastic model and 0.03 nats worse for SHELLELASTIC. Weight decay transfers as well (Figure 13 in Appendix I): under $\mu$-ELASTIC, the mean loss of SHELLELASTIC is flat over the same range as the proxy's, from $\lambda_0 = 10^{-3}$ to $10^{-1}$ (within 0.003 nats), whereas SP is higher at every swept value.

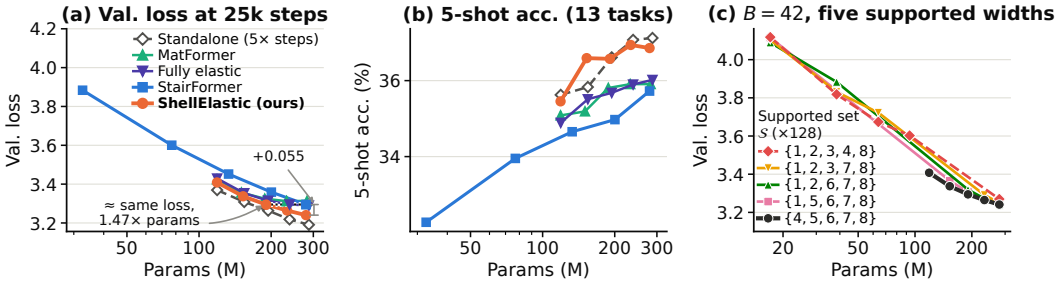


Figure 3: Architecture comparison and search against parameters at step 25,000. (a, b) Comparison with StairFormer: validation loss and average 5-shot accuracy (%) over 13 tasks. (c) Architecture search: validation loss of each supported width. Full results are in Appendices J.1 and J.2.

**Simultaneous maximal updates of $\mu$-ELASTIC.** Without width-dependent scaling, re-tuning one global value only trades one width against another (Proposition 2). Table 3 tests this limitation, giving SP the best of its 12 configurations in Figure 2 and $\mu$-ELASTIC only the base values. For the fully elastic model and SHELLELASTIC, $\mu$-ELASTIC still gives lower training and validation losses at every tier, by 0.012 and 0.013 nats in mean validation loss, respectively, as it keeps all widths maximally updated at once (Theorem 1). On MatFormer, $\mu$-ELASTIC is only slightly better (0.002 nats), likely because only its FFN down projection has a varying fan-in, so there is little for $\mu$-ELASTIC to correct across sub-networks. Under tuned SP, SHELLELASTIC still beats both baselines by 0.026 to 0.027 nats, so its lead is not due to $\mu$-ELASTIC alone.

### 4.3 ARCHITECTURE COMPARISON AND SEARCH

**Comparison with StairFormer.** StairFormer shares the recipe and HPs of SHELLELASTIC, including $\mu$-ELASTIC and all sub-networks at every step, and differs only in architecture: five equal 208-wide groups ($d_{\text{head}} = 104$), cumulative prefix normalization and one shared readout. With equal groups, its narrower sub-networks are much smaller (32.6M to 200.4M), and only its Max matches SHELLELASTIC in parameters (279.6M). At the parameter count of each SHELLELASTIC sub-network, its validation loss is 0.055 to 0.082 nats higher (Figure 3a; curves in Figure 15), and even its Max, with 1.47× the parameters, is 0.001 nats above the High of SHELLELASTIC. At Max, SHELLELASTIC also leads in average 5-shot accuracy, 36.9 against 35.7 (Figure 3b and Table 27). StairFormer starts from a single 208-wide block, whereas SHELLELASTIC starts from a 512-wide core, where the architecture search below finds extra blocks help most.

**Architecture search at equal parameters and width count.** Since $\mathcal{S}$ fixes both the extractable widths and the parameter count (a hidden matrix keeps $B(\mathcal{S}) = \sum_k (s_k - s_{k-1}) s_k$ of its $M^2$ blocks of size $g \times g$), we ask where extra width should go when $B$ and the number of supported widths are fixed. At $B = 42$ with five supported widths, each candidate set has four groups of width $g$ and one of width $4g$ and differs only in where this wide group sits (block masks in Figure 16); at $B = 39$ with six, the wide group has width $3g$ (Appendix J.2). We train every supported width at every step and compare the sets at each width they share with equal parameter counts (Figure 3c). The core is the best place for the wide group: the core placement has the lowest loss, or is within 0.006 nats of it, at every shared width, whereas a shell placement makes the first width containing the wide group 0.025 to 0.052 nats worse. This motivates the wide core and thin shells of SHELLELASTIC.

### 4.4 ABLATION STUDIES

Appendix K ablates the recipe one setting at a time: training every width at every step with loss weights $(1, 1, 1, 1, 4)$ is close to the best trade-off we found, separate terminal norms and readouts per width help, and no other change lowers the mean loss by more than 0.010 nats.

## 5 CONCLUSION

In SHELLELASTIC, one full-width forward pass contains every sub-network's exact hidden states as prefixes, and at equal parameters and width count a wide core with thin shells works best. With $\mu$-ELASTIC and HPs from a non-elastic proxy, one run yields multiple maximally updated models with downstream performance similar to standalone models trained with several times the steps.

### AI USE STATEMENT

In this work, we used generative AI tools to provide critical ingredients for proving our mathematical claims, to assist in writing the proofs (Appendices C and D), and to implement parts of our method. Additionally, we used generative AI tools to draft parts of the paper, edit it for readability, generate and check the code for the result figures and tables, and cross-check the statements of the paper against the training code. We have reviewed all AI-assisted work: we verified all AI-implemented code, checked every proof, and verified every number against the experiment logs. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work studies training methods for language models on publicly available web text and does not involve human subjects. Elastic models reduce the cost of serving several model sizes, which lowers the energy cost of deployment; they inherit the biases of their training data like any language model.

### REPRODUCIBILITY STATEMENT

Section 4.1 and Appendices F and G list every architectural, optimization and evaluation setting; Table 1 and Appendix E give the complete parameterization rules; Appendices C and D contain the proofs, and Appendix C.3 verifies Proposition 1 numerically; Appendices H, I, J and K give the complete main, transfer, architecture-search and ablation results. The training code, configurations and run logs used for all tables will be released.

## REFERENCES

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.

Blake Bordelon, Lorenzo Noci, Mufan Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. In *International Conference on Learning Representations*, volume 2024, pp. 22088–22127, 2024.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5298–5311. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/cai24e.html.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Fnu Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit S Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham M Kakade, Ali Farhadi, et al. Matformer: Nested transformer for elastic inference. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Nolan Dey, Bin Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *Advances in Neural Information Processing Systems*, 38:137707–137739, 2026.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 1286–1305, 2021.

Katie E Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/everett24a.html.

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

Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. Semeval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 394–398, 2012.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33: 9782–9793, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

Bruno Mlodozeniec, Pierre Ablin, Louis Béthune, Dan Busbridge, Michal Klein, Jason Ramapuram, et al. Completed hyperparameter transfer across modules, width, depth, batch and duration. In *International Conference on Learning Representations*, volume 2026, pp. 45300–45325, 2026.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Jascha Sohl-Dickstein, Roman Novak, Samuel S Schoenholz, and Jaehoon Lee. On the infinite width limit of neural networks with a standard parameterization. *arXiv preprint arXiv:2001.07301*, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ali Taghibakhshi, Sharath Turuvekere Sreenivas, Saurav Muralidharan, Ruisi Cai, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Yoshi Suhara, Oluwatobi Olabiyi, Daniel Korzekwa, Mostofa Patwary, et al. Nemotron elastic: Towards efficient many-in-one reasoning llms. *arXiv preprint arXiv:2511.16664*, 2025.

Avi Trost, Alexander Yun, John Cooper, Gabriel Orlanski, and Frederic Sala. Fully nested transformers. In *AdaptFM: Resource-Adaptive Foundation Model Inference*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite depth neural networks. In *International Conference on Learning Representations*, volume 2024, pp. 55099–55150, 2024.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the association for computational linguistics: NAACL 2024*, pp. 2299–2314, 2024.

## APPENDIX CONTENTS

**A Comprehensive related work** **15**

**B Architecture details** **15**

**C Exact computation reuse** **16**

C.1 Prefix-preserving maps . . . . . 16

C.2 Proof of Proposition 1 . . . . . 17

C.3 Numerical verification and scope . . . . . 19

**D Proofs of the $\mu$-ELASTIC results** **19**

D.1 Assumptions and notation . . . . . 19

D.2 Proof of Theorem 1 . . . . . 20

D.3 Proof of Proposition 2 . . . . . 21

D.4 Proof of Corollary 1 . . . . . 21

D.5 Extension to other elastic architectures . . . . . 21

**E $\mu$-ELASTIC for other elastic architectures** **22**

**F Training details** **23**

F.1 Model configurations . . . . . 24

F.2 Optimization . . . . . 24

F.3 Parameter counts . . . . . 24

F.4 Training and inference cost . . . . . 25

**G Evaluation details** **27**

**H Complete main results** **29**

**I Additional transfer results** **41**

**J Architecture comparison and search** **42**

J.1 Comparison with StairFormer . . . . . 42

J.2 Architecture search at equal parameters and width count . . . . . 45

**K Ablation studies** **46**

# A Comprehensive related work

**Elastic models.** Elastic models train several nested sub-networks in one shared parameter set, so one training run serves many inference budgets (Yu et al., 2018; Cai et al., 2020; Devvrit et al., 2024). Slimmable networks train one convolutional network at several widths (Yu et al., 2018; Yu & Huang, 2019), Once-for-All also nests depth, kernel size and resolution (Cai et al., 2020), and the same idea has been applied to the width and depth of BERT (Devlin et al., 2019; Hou et al., 2020) and to output representations (Kusupati et al., 2022); these recipes train the sub-networks jointly or by sampling, usually with distillation from the largest one (Hinton et al., 2015). For decoder-only transformers (Vaswani et al., 2017), MatFormer (Devvrit et al., 2024) nests only the FFN hidden width and underlies released on-device models (Gemma Team, 2025), while Flextron and Nemotron Elastic extract nested sub-networks from an already pretrained model (Cai et al., 2024; Taghibakhshi et al., 2025). The concurrent StairFormer (Trost et al., 2026) nests every smaller model as a prefix of one forward pass and instantiates this nesting with equal-width blocks; in its experiments, it trails dense models of the same size at every budget but supports a wider range of usable budgets than MatFormer. ShellElastic uses the same prefix structure but chooses the supported widths, which set the core and shell sizes, and normalizes within groups instead of over cumulative prefixes. None of the existing works derives how HPs should scale across nested widths or keeps every sub-network maximally updated.

**HP transfer and $\mu$-parameterization.** Tuning HPs such as the learning rate and initialization is expensive at scale, and under the standard parameterization (Sohl-Dickstein et al., 2020) their optima drift with width, so a small model cannot stand in for a large one (Yang et al., 2021). A parameterization that scales HPs with model size removes this drift, so values tuned on a small proxy transfer zero-shot (Yang et al., 2021; Dey et al., 2026). Building on infinite-width theory (Jacot et al., 2018; Mei et al., 2018), Yang & Hu (2021) identified $\mu$P, a parameterization that gives every layer maximal $\Theta(1)$ feature updates as width grows, and Yang et al. (2021) turned it into the $\mu$Transfer rule for initialization, learning rate and multipliers. CompleteP extends the rule across depth for pre-norm transformers (Yang et al., 2024; Bordelon et al., 2024; Dey et al., 2026), and Mlodozeniec et al. (2026) add batch size and token budget. Everett et al. (2024) derive per-layer learning-rate exponents across parameterizations and optimizers, including Adam, and find that HP transfer is not limited to $\mu$P; Table 1 follows their scaling rules. These rules assume separately trained models, where each tensor has a single fan-in; $\mu$-Elastic extends them to the shared, jointly trained tensors of an elastic supernet.

# B Architecture details

Figure 4 shows the complete ShellElastic model, Figure 5 details its attention and FFN sub-layers, and Figure 6 places the three elastic families side by side. Weight matrices are drawn with output rows and input columns and colored only where a connection is allowed; white lines mark the supported widths, and black lines delimit the slices of a dense matrix that the narrower sub-networks use. All three families are drawn with $\mathcal{S} = \{4, \ldots, 8\}$, the default supported set of ShellElastic, so that their matrices are directly comparable; Table 6 lists the dimensions of the trained models. The forward multipliers of Table 1 are omitted throughout, and the per-head query/key normalization and rotary position transformations appear only in Figure 5.

**Comparison with other elastic architectures.** MatFormer keeps the model width, attention, normalization, embedding and readout at full width and nests only the FFN hidden width. A fully elastic model slices dense matrices along both axes and normalizes over the active width. In both cases, channel mixing is dense, so the hidden states of a narrow sub-network are not contained in those of a wide one, and each width needs its own trunk pass. ShellElastic replaces the dense matrices with block-lower-triangular ones and the full-width normalization with group-local normalization, which is exactly what lets the supported prefixes share a single trunk pass (Proposition 1).

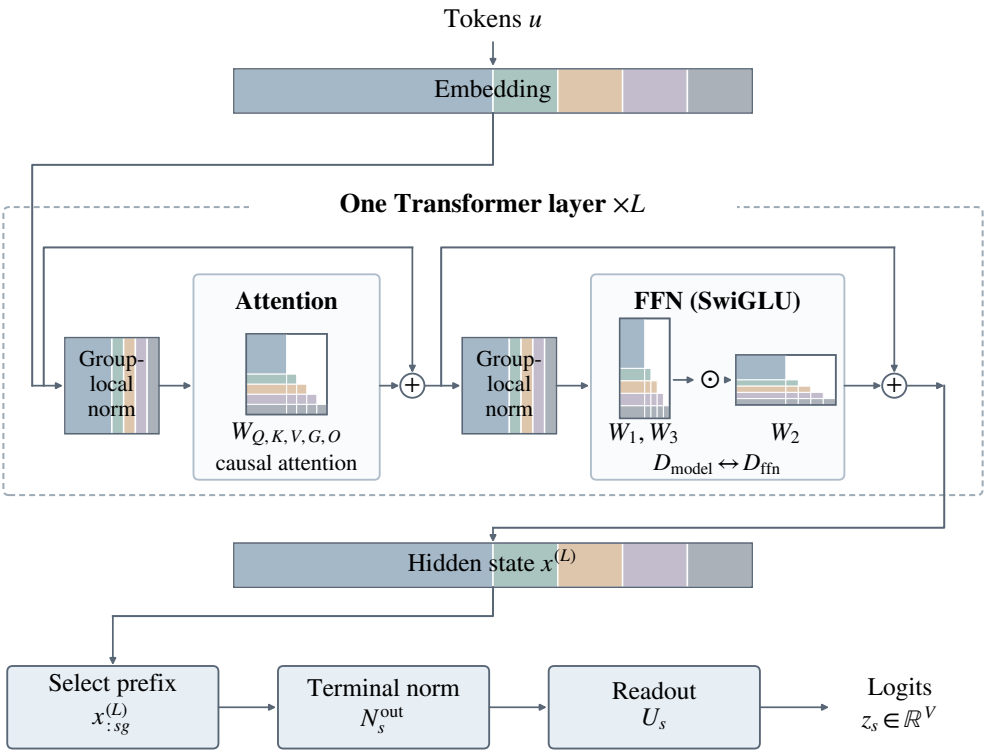


Figure 4: **The complete SHELLELASTIC model.** Embedding, one attention–FFN layer repeated $L$ times, and the terminal norm and readout of the selected width. Group-local branch normalization and structured projections preserve every supported prefix through the trunk.

## C EXACT COMPUTATION REUSE

This appendix proves Proposition 1. Appendix C.1 defines prefix-preserving maps and shows that the structured projections are prefix-preserving at the supported widths, Appendix C.2 composes the operations of the trunk, and Appendix C.3 checks the result numerically and delimits what it does and does not claim.

### C.1 PREFIX-PRESERVING MAPS

Throughout this appendix, $s \le t$ are supported widths in $\mathcal{S}$, and $P_s$ keeps the first $sg$ coordinates of every token's hidden state (the first $esg$ coordinates on the FFN hidden axis), as in Proposition 1. For a map $F$ that the trunk applies at every supported width, $F_s$ denotes its restriction to width $sg$, that is, the map that sub-network $s$ applies.

**Definition 1** (Prefix-preserving map)**.** *A map $F$ is* prefix-preserving *if*

$$P_s \circ F_t = F_s \circ P_s \qquad \text{for all } s, t \in \mathcal{S},\ s \le t. \tag{10}$$

We first show that the structured projections of Section 3.1 have this property. The lemma allows different base granularities on the two axes, since the output groups of the FFN up and gate projections are $e$ times as wide as their input groups, and those of the down projection $e$ times as narrow.

**Lemma 1** (Structured linear maps)**.** *Consider a linear map with base input and output granularities $g_{\text{in}}$ and $g_{\text{out}}$, in which output group $k$ has $(s_k - s_{k-1})g_{\text{out}}$ rows and reads the first $s_k g_{\text{in}}$ input coordinates. Let $P_s^{\text{in}}$ and $P_s^{\text{out}}$ keep the first $sg_{\text{in}}$ inputs and the first $sg_{\text{out}}$ outputs, and let $W_s$ be the top-left $sg_{\text{out}} \times sg_{\text{in}}$ block of the weight matrix. Then, with the row-vector convention,*

$$P_s^{\text{out}}\left(xW_t^\top\right) = P_s^{\text{in}}(x)\, W_s^\top, \qquad s, t \in \mathcal{S}, \quad s \le t. \tag{11}$$

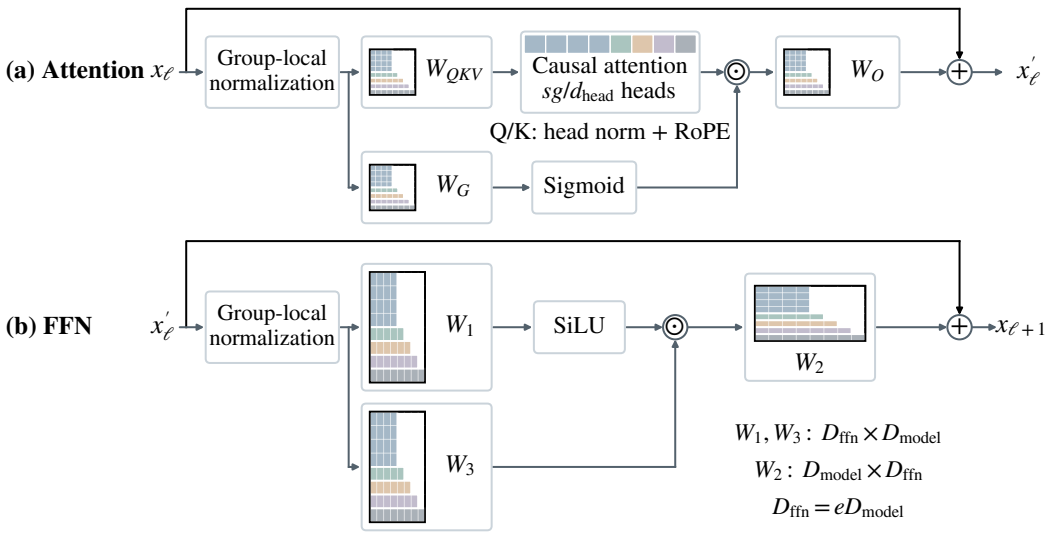


Figure 5: **Prefix-preserving attention and FFN.** Sub-network $s$ uses the first $sg/d_{\text{head}}$ attention heads. The FFN's up, gate, and down projections expand and contract the same groups.

*Proof.* Let $s = s_j$. The first $sg_{\text{out}}$ outputs are those of the groups $k \le j$, and output coordinate $i$ of group $k$ is

$$\left(xW_t^\top\right)_i = \sum_{c=1}^{s_k g_{\text{in}}} (W_t)_{ic}\, x_c. \tag{12}$$

Since $k \le j$ gives $s_k \le s_j = s$, this sum reads only the first $sg_{\text{in}}$ inputs, and all of its weights lie in the top-left block $W_s$. Hence it equals $\left(P_s^{\text{in}}(x)\, W_s^\top\right)_i$. □

**Unsupported widths.** At an index $u \notin \mathcal{S}$, there is a group with $s_{k-1} < u < s_k$. A cut at $u$ keeps part of that group but removes inputs it is allowed to read (Equation 12), so Equation 11 does not hold in general, and the architecture guarantees no reusable sub-network there. For $\mathcal{S} = \{4, 5, 6, 7, 8\}$, the indices $1, 2, 3$ lie inside the dense core and are not supported.

### C.2 Proof of Proposition 1

*Proof.* We show that every operation of a layer is prefix-preserving (Steps 1–4) and then compose the operations over the $L$ layers (Step 5).

**Step 1: projections.** The query, key, value, gate and output projections of attention and the up, gate and down projections of the FFN are structured linear maps. Lemma 1 therefore applies to each of them, with $g_{\text{in}} = g_{\text{out}} = g$ for the attention projections, $(g_{\text{in}}, g_{\text{out}}) = (g, eg)$ for the FFN up and gate projections, and $(eg, g)$ for the FFN down projection.

**Step 2: group-local normalization.** By Equation 3, $N(x)_{\mathcal{G}_k} = N_k(x_{\mathcal{G}_k})$ uses only the coordinates and gains of $\mathcal{G}_k$, so the first $sg$ outputs depend only on the first $sg$ inputs, and $P_s \circ N_t = N_s \circ P_s$. The same holds for any finer partition and for a cumulative prefix norm, whose statistics for a coordinate use all coordinates up to the end of its group or of its block of $g$ coordinates (Equation 25); for any of the first $sg$ coordinates, that end is at most $sg$. A norm over the full active width does not preserve prefixes, because adding a group changes the statistics of the earlier coordinates.

**Step 3: attention.** Since $g$ is a multiple of $d_{\text{head}}$, every head lies inside one group, and sub-network $s$ uses the first $sg/d_{\text{head}}$ heads. By Step 1, their queries, keys and values depend only on the first $sg$ inputs. Query–key normalization, the rotary transformation and the softmax act within a head, so the heads that sub-network $t$ adds leave the outputs of these heads unchanged. The sigmoid gate and the output projection of Equation 4 are covered by Steps 1 and 4.

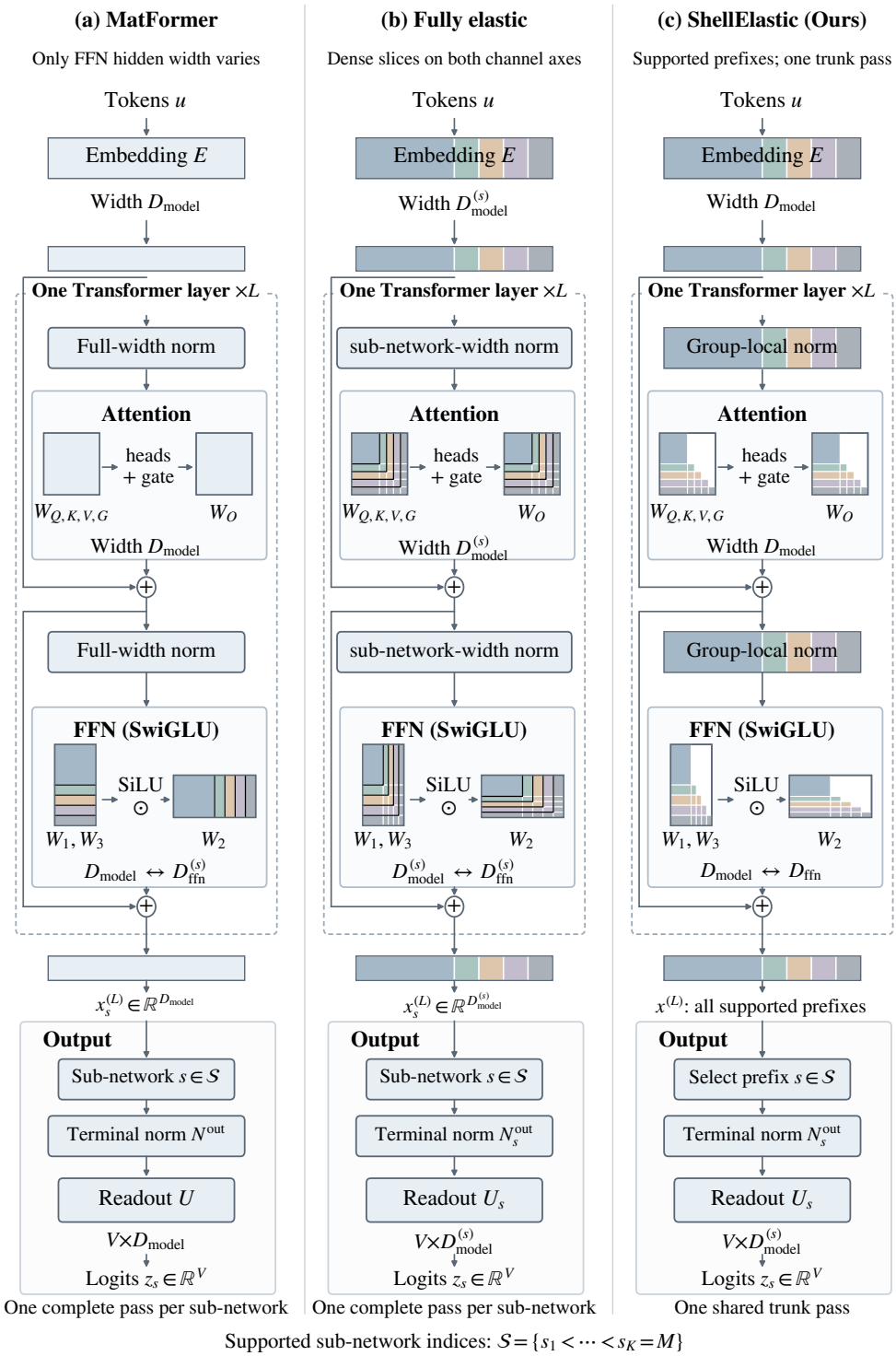


Figure 6: **Complete architecture comparison.** MatFormer varies the FFN hidden dimension; fully elastic models vary the model dimension and the FFN hidden dimension together by slicing dense matrices; SHELLELASTIC also varies both, but through block-lower-triangular projections in which each output group reads only a prefix of the input. Each layer frame repeats $L$ times. Proportions are illustrative.

**Step 4: elementwise operations, residuals and embedding.** The SwiGLU product and the other elementwise operations act coordinatewise, so they commute with $P_s$. A residual branch $x \mapsto x+F(x)$ with a prefix-preserving $F$ is prefix-preserving, since $P_s\big(x+F_t(x)\big) = P_s(x)+F_s\big(P_s(x)\big)$. The embedding satisfies $P_s(E_t[u]) = E_s[u]$ by construction.

**Step 5: composition over layers.** If $F$ and $G$ are prefix-preserving, then so is $G \circ F$:

$$P_s \circ G_t \circ F_t = G_s \circ P_s \circ F_t = G_s \circ F_s \circ P_s. \tag{13}$$

Each layer is a composition of the operations of Steps 1–4 and is therefore prefix-preserving, and by induction over the $L$ layers, as in Proposition 2.3 of StairFormer (Trost et al., 2026), so is the trunk: $P_s \circ T_t = T_s \circ P_s$. Applying this identity to the embedded sequence and using Step 4,

$$P_s\big(T_t(E_t[u])\big) = T_s\big(P_s(E_t[u])\big) = T_s\big(E_s[u]\big), \tag{14}$$

which is the claim. □

**Consequence for the joint objective.** The terminal norm $N_s^{\text{out}}$ and the readout $U_s$ need not commute with $P_s$, since $N_s^{\text{out}}$ normalizes over all $sg$ selected coordinates. They are applied *after* the prefix has been selected (Equation 6), so each sub-network's terminal map receives exactly the input it would receive in a separate evaluation. Hence each $\ell_s$, and the gradient of the joint objective of Equation 8, is the same function of the shared parameters as in separate evaluations, up to floating-point summation order.

### C.3 Numerical verification and scope

**Numerical verification.** We instantiated the full trained configuration on the meta device to check module selection and shapes, and ran a reduced two-layer model on CPU with the same architecture switches, comparing one shared trunk pass against five separate evaluations of the same nested weights. The logits and the weighted cross-entropy agreed exactly; the largest absolute difference between the two sets of parameter gradients was $1.68 \times 10^{-8}$, and every trainable parameter received a gradient. The check replaces the fused SwiGLU and cross-entropy kernels of the GPU path with reference expressions, so it supports the implementation of Proposition 1, not its throughput.

**Scope of the statement.** The proposition assumes the dense-FFN graph of Section 3.1, a fixed supported set $\mathcal{S}$ and prefix-preserving normalization. Beyond these assumptions:

- dropout needs matched masks for an equality test;
- data-dependent routing or width-specific trunk modules need a separate argument;
- Equation 2 fixes the logical dependencies and not a storage layout or a kernel: every implementation of that map has the prefix property.

## D Proofs of the $\mu$-Elastic results

This appendix proves Theorem 1, Proposition 2 and Corollary 1. Appendix D.1 states the assumptions and fixes the notation, Appendices D.2–D.4 give the three proofs, and Appendix D.5 extends Theorem 1 and Corollary 1 to the other elastic architectures of Section 3.2.

### D.1 Assumptions and notation

**Assumption 1** (Heuristic scaling)**.** *We use the heuristic framework of Appendix J of the $\mu$Transfer paper (Yang et al., 2021):*

*(A1)* Inputs. *The input to every hidden matrix has $\Theta(1)$ coordinates with bounded second moments and is independent of that matrix at initialization.*

*(A2)* Central limit. *If $w_1, \dots, w_n$ are i.i.d. with mean zero and standard deviation $b$ and independent of $x_1, \dots, x_n = \Theta(1)$, then $\sum_{c=1}^{n} w_c x_c = \Theta(\sqrt{n}\, b)$.*

*(A3)* Law of large numbers. *After a gradient step, the update $\Delta W$ of a hidden matrix is aligned with the activations it multiplies, so $\sum_{c=1}^{n} \Delta W_{jc}\, x_c = \Theta(n\,\delta)$ when $|\Delta W_{jc}| = \Theta(\delta)$.*

*(A4)* Adam. *With learning rate $\rho$ and a non-degenerate second-moment estimate, every coordinate of an Adam update has magnitude $\Theta(\rho)$, independently of the gradient scale.*

**Notation.** Consider a hidden matrix and the block row $W_k \in \mathbb{R}^{|\mathcal{G}_k| \times sg}$ of Equation 2 that produces output group $\mathcal{G}_k$, whose supported boundary is $s = s_k$. Inside a sub-network $t \in \mathcal{S}$ with $t \geq s$, let $x$ be the input of the matrix, and let

$$y_j = \sum_{c=1}^{sg} (W_k)_{jc}\, x_c, \qquad j \in \mathcal{G}_k, \tag{15}$$

be the pre-activation of coordinate $j$. After one optimizer step, the update $\Delta W_k$ moves it by $\Delta y_j = \sum_{c=1}^{sg} (\Delta W_k)_{jc}\, x_c$. Read widths are measured in base widths: the block row of $\mathcal{G}_k$ reads $s_k = s$ of them, and the readout of sub-network $s$ reads $s$. The proxy width $d_0$ and the base values $(\sigma_0, \eta_0, \lambda_0)$ are fixed, and the constants hidden in $\Theta(\cdot)$ do not depend on $g$, $s$, $t$, $M$, $m$ or $e$.

### D.2 Proof of Theorem 1

*Proof.* **Step 1: the fan-in does not depend on the sub-network.** The sum in Equation 15 has exactly $sg$ terms in every sub-network $t \geq s$: the block row reads no coordinate beyond $sg$ (Equation 2), and sub-network $t$ contains all of them because $tg \geq sg$. One initialization scale and one learning rate, set by $q(s) = ms = sg/d_0$ (Equation 7), therefore serve every sub-network that contains the group.

**Step 2: initialization.** Table 1 draws the entries of $W_k$ i.i.d. with standard deviation $\sigma_0/\sqrt{ms} = \sigma_0\sqrt{d_0/(sg)}$. By (A1) and (A2),

$$\mathbb{E}\big[y_j^2\big] = sg \cdot \frac{\sigma_0^2 d_0}{sg} \cdot \Theta(1) = \Theta\big(\sigma_0^2 d_0\big)\,. \tag{16}$$

**Step 3: one AdamW step.** The learning rate of $W_k$ is $\eta_0/(ms) = \eta_0 d_0/(sg)$, so by (A4) every coordinate of its update satisfies $|(\Delta W_k)_{jc}| = \Theta(\eta_0 d_0/(sg))$, and by (A3)

$$\Delta y_j = \sum_{c=1}^{sg} (\Delta W_k)_{jc}\, x_c = \Theta\left(sg \cdot \frac{\eta_0 d_0}{sg}\right) = \Theta(\eta_0 d_0)\,. \tag{17}$$

The decoupled weight decay of AdamW, which is scaled by the learning rate, multiplies each coordinate by

$$1 - \frac{\eta_0}{ms} \cdot \lambda_0 ms = 1 - \eta_0\lambda_0 \tag{18}$$

per step, the same factor for every decayed parameter group and every width.

**Step 4: FFN down projection.** In the FFN down projection, the block row of $\mathcal{G}_k$ reads $esg$ coordinates, and its output carries the forward multiplier $1/e$ (Table 1). With initialization $\sigma_0\sqrt{e}/\sqrt{ms}$ and learning rate $\eta_0/(ms)$, the computations of Steps 2 and 3 become

$$\mathbb{E}\big[y_j^2\big] = e^{-2} \cdot esg \cdot \frac{e\,\sigma_0^2 d_0}{sg} \cdot \Theta(1) = \Theta\big(\sigma_0^2 d_0\big)\,, \tag{19}$$

$$\Delta y_j = \Theta\left(e^{-1} \cdot esg \cdot \frac{\eta_0 d_0}{sg}\right) = \Theta(\eta_0 d_0)\,, \tag{20}$$

so both scales are also free of $e$.

**Step 5: readouts and embedding.** The readout $U_s$ reads the $sg$ coordinates of $N_s^{\mathrm{out}}(x_{:sg}^{(L)})$ (Equation 6), which is exactly the fan-in of the group with boundary $s$, and Table 1 gives it the same initialization and learning rate, following the per-layer scaling of Everett et al. (2024). Steps 2 and 3 therefore give the same two bounds for its logits at initialization and for their change under the update of $U_s$, and the learned scalar $\alpha_s$ rescales them only by $\Theta(1)$. The embedding is a lookup over a fixed vocabulary of size $V$, which does not change with width, and reading its first $sg$ columns does not change the scale of any coordinate.

**Step 6: joint training.** Under the joint objective of Equation 8, the gradient of $W_k$ sums the contributions of the selected sub-networks $t \geq s$. By Proposition 1, all of them feed $W_k$ the same input $x$, so every contribution is an outer product with $x$, and the summed gradient, whatever its scale, keeps each row aligned with $x$; (A3) therefore applies, and by (A4) the Adam update has coordinates of size $\Theta(\eta_0/(ms))$. The group receives a gradient whenever some selected $t \geq s$ has $\omega_t > 0$, which holds for $t = M$ when $\omega_M > 0$, and the readout $U_s$ whenever $s \in \mathcal{A}_\tau$ and $\omega_s > 0$. Steps 2–5 therefore hold for the jointly trained supernet.

The constants $\Theta\big(\sigma_0^2 d_0\big)$ and $\Theta(\eta_0 d_0)$ of Equations 16 and 17 depend on neither $s$, $t$, $M$ nor $m$, which proves (i) and (ii) for every hidden group and every readout. □

### D.3 Proof of Proposition 2

*Proof.* Consider a hidden matrix whose input satisfies (A1), such as a projection that reads a group-local norm, and let all of its groups share the initialization standard deviation $b$, the learning rate $\rho$ and the forward multiplier, which we absorb into $b$ and $\rho$.

**Fixed ratios.** The group with boundary $s$ sums over $sg$ coordinates (Step 1 of Appendix D.2), so by (A2)–(A4)

$$y_j = \Theta(\sqrt{sg}\, b)\,, \qquad \Delta y_j = \Theta(sg\, \rho)\,. \tag{21}$$

For boundaries $s < s'$, the pre-activations and updates of the two groups therefore differ by the factors $\Theta\Big(\sqrt{s'/s}\Big)$ and $\Theta(s'/s)$, which do not depend on $(b, \rho)$: changing the shared pair rescales all groups together and leaves these factors unchanged.

**Calibrations.** Calibrating the shared pair for the group with boundary $r \in \mathcal{S}$ means taking the values of Table 1 for that group, $b = \sigma_0\sqrt{d_0/(rg)}$ and $\rho = \eta_0 d_0/(rg)$. By Equation 21, the group with boundary $s$ then has

$$y_j = \Theta\Big(\sigma_0\sqrt{d_0}\cdot\sqrt{s/r}\Big)\,, \qquad \Delta y_j = \Theta(\eta_0 d_0 \cdot s/r)\,, \tag{22}$$

so it deviates from the scales $\Theta\big(\sigma_0\sqrt{d_0}\big)$ and $\Theta(\eta_0 d_0)$ of Theorem 1 by the factors $\sqrt{s/r}$ and $s/r$. For $r = M$, the group with boundary $s$ has pre-activations of size $\Theta\Big(\sqrt{s/M}\Big)$ and updates of size $\Theta(\eta_0\, s/M)$, up to the constants $\sigma_0\sqrt{d_0}$ and $d_0$. For $r = s_1$, it is inflated by $\sqrt{s/s_1}$ and $s/s_1$, and the outermost group by $\sqrt{M/s_1}$ and $M/s_1$.

**No shared pair is uniformly maximal.** Let the core and the outermost group deviate from the scales of Theorem 1 by the factors $a_1$ and $a_K$ at initialization. By the fixed ratios, $a_K/a_1 = \Theta\Big(\sqrt{M/s_1}\Big)$, so $\max(a_1, a_1^{-1}, a_K, a_K^{-1})$ grows at least as $(M/s_1)^{1/4}$; the same argument for the updates gives $(M/s_1)^{1/2}$. Over supernets with unbounded $M/s_1$, no shared pair keeps both factors within constants independent of $M$, so none is maximal in the sense of Theorem 1 for every group. When $M/s_1$ is bounded, the mismatch is a bounded factor: for $\mathcal{S} = \{4, \ldots, 8\}$, $M/s_1 = 2$, and any shared pair leaves the core and the outermost group $\sqrt{2}$ apart at initialization and 2 apart in the update. □

### D.4 Proof of Corollary 1

*Proof.* The proof of Theorem 1 holds for every base width $g = md_0$ and every supported set $\mathcal{S} \subseteq \{1, \ldots, M\}$. Its constants, $\Theta\big(\sigma_0^2 d_0\big)$ in Equation 16, $\Theta(\eta_0 d_0)$ in Equation 17 and the decay factor $1 - \eta_0\lambda_0$ in Equation 18, depend only on the proxy width $d_0$ and on the tuned values: $m$ and $M$ enter only through the active input widths, which the rule of Table 1 cancels. The proxy is the special case $M = 1$, $g = d_0$ (so $m = s = 1$) of the same computation, so the values tuned on it set the same constants in every group and readout of every target. This gives (i) for any ShellElastic supernet and (ii) under any increase of $m$ or $M$. □

### D.5 Extension to other elastic architectures

The proofs above use the block structure of ShellElastic only in Step 1 of Appendix D.2, to show that a group has the same fan-in and the same input in every sub-network that contains it. For

the other elastic architectures of Section 3.2, under their rules in Appendix E, Theorem 1 holds with sub-network $s$ in place of the group with boundary $s$: the hidden matrices of every sub-network satisfy (i), and (ii) at each step that trains that sub-network. The readouts are treated per architecture below.

**StairFormer.** Its hidden matrices have the connectivity of SHELLELASTIC with $\mathcal{S} = \{1, \dots, M\}$ and follow Table 1, and all sub-networks are trained at every step, so the proof of Theorem 1 applies verbatim. Its single readout, of which sub-network $s$ reads the first $sg$ columns, takes the split of Equation 23, which gives these columns the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of $U_s$ in Table 1; since every sub-network feeds them the prefix $x^{(L)}_{:sg}$ up to its terminal norm, Steps 5 and 6 apply.

**Fully elastic models.** Sub-network $s$ reads the top-left slice $W_s$ of every hidden matrix, whose fan-in $sg$ ($esg$ for the FFN down projection) varies with $s$. By Equation 23, $W_s$ has the initialization standard deviation $\sigma_0/\sqrt{ms}$ ($\sigma_0\sqrt{e}/\sqrt{ms}$ with the output multiplier $1/e$ for the FFN down projection), and since $\Delta$ moves by $\Theta(\eta_0/m)$ per coordinate under (A4), $W_s$ moves by $\Theta(\eta_0/(ms))$ per coordinate. These are the values of Table 1 at fan-in $sg$, so Steps 2–4 give the pre-activations of sub-network $s$ the scales $\Theta\big(\sigma_0\sqrt{d_0}\big)$ at initialization and $\Theta(\eta_0 d_0)$ after one step. At a step that trains sub-network $s$, the update of $\Delta$ is computed from the input of sub-network $s$, so (A3) applies to the update of $W_s$; (ii) does not need the steps that train other sub-networks. The shared readout takes the same split with the multiplier $\alpha_s$ (Appendix E), so the columns that sub-network $s$ reads have the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of $U_s$ in Table 1, and Step 5 gives its logits the same two scales.

**MatFormer.** Only the FFN down projection has a fan-in that varies with the sub-network, $sh_0 = esD_{\text{model}} = esmd_0$. With the split and the output multiplier $1/e$, its slice has the effective initialization $\sigma_0/\sqrt{ems}$ and the effective learning rate $\eta_0/(ems)$, so Step 4 gives the same constants with $sD_{\text{model}}$ in place of $sg$. Every other hidden matrix and the shared readout have the fixed fan-in $D_{\text{model}} = md_0$ and take the factor $m$, which is the rule of Table 1 with $s = 1$ and gives the same constants by Steps 2, 3 and 5. As for fully elastic models, (A3) applies at each step to the sub-network trained at that step.

**Transfer.** In all three architectures, $m$ and $s$ enter only through fan-ins that the rules cancel, so the constants $\Theta\big(\sigma_0\sqrt{d_0}\big)$ and $\Theta(\eta_0 d_0)$ and the decay factor $1 - \eta_0\lambda_0$, which $\Delta$ obtains from its learning rate $\eta_0/m$ and weight decay $\lambda_0 m$, depend only on $d_0$ and the base values. The proof of Corollary 1 therefore carries over to every $m$ and every supported set.

## E $\mu$-ELASTIC FOR OTHER ELASTIC ARCHITECTURES

All three elastic baselines use the proxy width $d_0 = 128$ and the base values $(\sigma_0, \eta_0, \lambda_0)$ of SHELLELASTIC, and their $m$ and $s$ follow Section 3.2. Tables 4 and 5 list the complete rules of the fully elastic model and MatFormer.

**Fully elastic models.** A fully elastic model stores every hidden matrix as one dense tensor at the full width, and sub-network $s$ reads its top-left slice, whose fan-in $sg$ ($esg$ for the FFN down projection) varies with $s$. $\mu$-ELASTIC gives this slice the factor $ms$: initialization $\sigma_0/\sqrt{ms}$, learning rate $\eta_0/(ms)$ and weight decay $\lambda_0 ms$. One stored tensor cannot follow this rule for every $s$, because the initialization and the update scale with different powers of $s$. We therefore split each hidden tensor into a frozen initialization $W^{\text{init}}$, drawn at $\sigma_0/\sqrt{m}$, and a trainable delta $\Delta$, initialized to zero and trained with learning rate $\eta_0/m$ and weight decay $\lambda_0 m$. Sub-network $s$ uses

$$W_s = s^{-1/2}\, W^{\text{init}}[\text{slice}] + s^{-1}\, \Delta[\text{slice}]. \tag{23}$$

At initialization, $W_s$ has standard deviation $s^{-1/2}\sigma_0/\sqrt{m} = \sigma_0/\sqrt{ms}$; under Adam, $\Delta$ moves by $\Theta(\eta_0/m)$ per coordinate, so $W_s$ moves by $\Theta(\eta_0/(ms))$. Both match Table 1 for every $s$, while weight decay, which acts on $\Delta$ only, pulls $W_s$ toward its initialization. The main results use one shared sliced readout, of which sub-network $s$ reads the first $sg$ columns; it takes the same split with the multiplier $\alpha_s$ (Table 4), which gives these columns the initialization $\sigma_0/\sqrt{ms}$ and the learning rate $\eta_0/(ms)$ of the per-width readout $U_s$ of Table 1, and a run with per-width readouts $U_s$ agrees within 0.001 nats. Our baseline has $g = 64$ ($m = 1/2$), $\mathcal{S} = \{6, \dots, 10\}$ and $e = 4$; attention

Table 4: **$\mu$-ELASTIC for the fully elastic baseline.** $s$ is the sub-network width in units of $g$; multipliers act on the sliced tensor in the forward pass (Equation 23), and the FFN down projection multiplies its output, from both terms, by $1/e$.

| Parameter | Initialization | Learning rate | Weight decay | Forward multiplier |
|---|---|---|---|---|
| Token embedding (first $sg$ columns) | $\sigma_0$ | $\eta_0$ | $\lambda_0$ | 1 |
| $W^{\text{init}}$ of attention and FFN up/gate | $\sigma_0/\sqrt{m}$ | frozen | – | $s^{-1/2}$ |
| $W^{\text{init}}$ of FFN down | $\sigma_0\sqrt{e}/\sqrt{m}$ | frozen | – | $s^{-1/2}$ |
| $\Delta$ of every hidden matrix | 0 | $\eta_0/m$ | $\lambda_0 m$ | $s^{-1}$ |
| $W^{\text{init}}$ of the readout (first $sg$ columns) | $\sigma_0/\sqrt{m}$ | frozen | – | $\alpha_s s^{-1/2}$ |
| $\Delta$ of the readout (first $sg$ columns) | 0 | $\eta_0/m$ | $\lambda_0 m$ | $\alpha_s s^{-1}$ |
| Norm gains, readout scalars $\alpha_s$ | 1 | $\eta_0$ | 0 | – |

Table 5: **$\mu$-ELASTIC for MatFormer.** $s = D_{\text{ffn}}^{(s)}/h_0$ is the FFN hidden width in units of the base FFN width $h_0$; the hidden matrices other than the FFN down projection and the readout have a fixed fan-in and take the non-elastic $\mu$P factor $m = D_{\text{model}}/d_0$, and the embedding and gains keep the base values. The FFN down projection multiplies its output, from both terms, by $1/e$.

| Parameter | Initialization | Learning rate | Weight decay | Forward multiplier |
|---|---|---|---|---|
| Token embedding (full width) | $\sigma_0$ | $\eta_0$ | $\lambda_0$ | 1 |
| $W^{\text{init}}$ of attention and FFN up/gate (rows sliced to $D_{\text{ffn}}^{(s)}$) | $\sigma_0/\sqrt{m}$ | frozen | – | 1 |
| $\Delta$ of attention and FFN up/gate | 0 | $\eta_0/m$ | $\lambda_0 m$ | 1 |
| $W^{\text{init}}$ of FFN down (columns sliced to $D_{\text{ffn}}^{(s)}$) | $\sigma_0\sqrt{e}/\sqrt{m}$ | frozen | – | $s^{-1/2}$ |
| $\Delta$ of FFN down | 0 | $\eta_0/m$ | $\lambda_0 m$ | $s^{-1}$ |
| Shared readout (fan-in $D_{\text{model}}$) | $\sigma_0/\sqrt{m}$ | $\eta_0/m$ | $\lambda_0 m$ | $\alpha$ (one scalar) |
| Norm gains, readout scalar $\alpha$ | 1 | $\eta_0$ | 0 | – |

nests by heads at $d_{\text{head}} = 64$, the norms act over the active width, the embedding is sliced, and one sub-network is sampled uniformly at random per step.

**MatFormer.** MatFormer keeps $D_{\text{model}}$ fixed, so $m = D_{\text{model}}/d_0$, and nests only the FFN hidden width, $D_{\text{ffn}}^{(s)} = s\,h_0$ with a base FFN width $h_0$; here $e = h_0/D_{\text{model}}$ takes the place of the FFN expansion, so that the down projection of sub-network $s$ reads $esD_{\text{model}}$ coordinates. Only the FFN down projection has a fan-in that varies across sub-networks; it takes the factor $ms$ through the split of Equation 23, and the other hidden matrices and the readout, whose fan-in $D_{\text{model}}$ is fixed, take the non-elastic $\mu$P factor $m$, the attention and FFN up/gate matrices through the same split with multiplier 1; the embedding and gains keep the base values (Table 5). Our baseline has $D_{\text{model}} = 512$ ($m = 4$), $h_0 = 128$ ($e = 1/4$) and $\mathcal{S} = \{4, 9, 15, 22, 30\}$; attention, normalization, the embedding and a single readout are shared at full width, and one sub-network is sampled uniformly at random per step, as in the original MatFormer recipe.

**StairFormer.** With equal-width nested blocks, StairFormer has the connectivity of SHELLELASTIC with $\mathcal{S} = \{1, \ldots, M\}$, so its hidden matrices follow Table 1 without a split. Its single readout, of which sub-network $s$ reads the first $sg$ columns, has a fan-in that varies with $s$ and takes the factor $ms$ through the split of Equation 23. Our StairFormer baseline has five 208-wide groups with $d_{\text{head}} = 104$, so that each group holds two complete heads.

## F TRAINING DETAILS

This appendix gives the model configurations of the three scales (Appendix F.1), the optimization settings (Appendix F.2), the parameter counts (Appendix F.3), and the training and inference cost (Appendix F.4).

### F.1 Model configurations

Table 6 lists the base width, supported set and size of every model, and Table 7 the remaining architecture settings; Figure 5 shows one layer of ShellElastic and Figure 4 the complete model. Ablations vary only the setting under study relative to the defaults of the 119M–285M scale.

**119M–285M scale.** The main results at this scale use the models of the first block of Table 6, trained on 6.55B DCLM-Baseline tokens with the HPs of Table 8. ShellElastic has $g = 128$ with one head per base width ($d_{\text{head}} = 128$), $M = 8$ and $\mathcal{S} = \{4, 5, 6, 7, 8\}$, so each hidden matrix stores 42 of its 64 blocks of size $128 \times 128$, and the FFN expansion is $e = 1$. The two branch norms of every layer are group-local over $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ of widths $(512, 128, 128, 128, 128)$, queries and keys are normalized per head, and each supported width has its own terminal norm $N_s^{\text{out}}$ over all $sg$ coordinates, readout $U_s$ and learned scalar $\alpha_s$; all five widths are selected at every step with loss weights $(1, 1, 1, 1, 4)$. The proxy is the non-elastic special case $M = 1$ at width $d_0 = 128$, and each standalone model is a separately trained model of width 384 to 640 with FFN expansion 4 and $d_{\text{head}} = 64$; the sampled standalone baseline trains these five models in one run of the default length, updating one model sampled uniformly at random per step.

**375M–906M scale.** The main results at this scale use the models of the second block of Table 6. ShellElastic has $g = 256$ with two heads per base width ($d_{\text{head}} = 128$, $m = 2$) and otherwise the settings above: $M = 8$, $\mathcal{S} = \{4, 5, 6, 7, 8\}$, $e = 1$, groups $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ of widths $(1024, 256, 256, 256, 256)$ and loss weights $(1, 1, 1, 1, 4)$. MatFormer has $D_{\text{model}} = 1024$ ($m = 8$), $h_0 = 128$ ($e = 1/8$) and $\mathcal{S} = \{8, 17, 27, 38, 51\}$, and the fully elastic model has $g = 64$ ($m = 1/2$), $\mathcal{S} = \{13, 15, 17, 19, 21\}$ and $e = 2.75$. The standalone models have widths 832 to 1344 ($g = 64$, $M = 13, 15, 17, 19, 21$) with FFN expansion 2.75 and $d_{\text{head}} = 64$, and the sampled standalone baseline trains these five models in one run. Every run at this scale takes 50,000 steps of 256 sequences (26.2B tokens), decays the learning rate linearly from step 40,000 to 0 at step 50,000 and reports the validation loss at step 50,000; all other settings are those of the 119M–285M scale.

**675M–1.69B scale.** The main results at this scale use the models of the third block of Table 6. ShellElastic is the supernet of the 375M–906M scale with FFN expansion $e = 4$. MatFormer has $D_{\text{model}} = 1024$ ($m = 8$), $h_0 = 256$ ($e = 1/4$) and $\mathcal{S} = \{16, 24, 34, 45, 57\}$, and the fully elastic model and the standalone models keep the widths of the 375M–906M scale with FFN expansion 7.25. Every run at this scale takes 100,000 steps of 256 sequences (52.4B tokens), decays the learning rate linearly from step 85,000 to 0 at step 100,000 and reports the validation loss at step 100,000. For the doubled horizon, the optimizer HPs are rescaled to $\eta_0 = 1.41 \times 10^{-3}$, $\lambda_0 = 7.07 \times 10^{-3}$, $\beta_1 = \beta_2 = 0.975$ and $\epsilon = 1.41 \times 10^{-14}$, and the sampled standalone baseline rescales its own values in the same way; each model is fully sharded over the 8 GPUs, and all other settings are those of the 375M–906M scale.

### F.2 Optimization

Table 8 lists the optimization settings used unless otherwise specified: the proxy sweeps, the elastic models, the baselines and the standalone models differ only in their architecture and in the per-group multipliers of their parameterization. The training objective includes the $z$-loss of Equation 8 with $\lambda_z = 10^{-4}$; all reported losses, both training and validation, are the cross-entropy alone, averaged over non-padded tokens.

### F.3 Parameter counts

The parameters of a sub-network count the non-zero weights it reads, its embedding columns, and its own terminal norm and readout; the two tensors of a frozen-plus-delta split (Equation 23) count once. The baselines were sized to these counts (Table 7), and Table 9 lists them for every sub-network of the main results at the three scales.

Table 6: **Model configurations.** Each model is defined by its base width $g$ and supported set $\mathcal{S}$ (Section 3.1): its sub-networks have widths $sg$ for $s \in \mathcal{S}$, and a non-elastic model has $\mathcal{S} = \{M\}$. MatFormer nests only the FFN hidden width, so its $g$ and $\mathcal{S}$ refer to that width at a fixed $D_{\text{model}}$ (512, and 1024 at the two larger scales). All models have 32 layers and a 50,432-token vocabulary; parameters count the non-zero weights of each sub-network (Appendix F.3).

| Model | $g$ | $\mathcal{S}$ | Sub-network widths | $d_{\text{head}}$ | $D_{\text{ffn}}$ | Parameters (M) |
|---|---|---|---|---|---|---|
| Proxy | 128 | $\{1\}$ | 128 | 128 | 128 | 17.1 |
| *119M–285M scale* | | | | | | |
| Standalone | 64 | $\{M\}$, $M = 6, 7, 8, 9, 10$ | 384–640 | 64 | $4D_{\text{model}}$ | 119.0–287.5 |
| MatFormer | 128 | $\{4, 9, 15, 22, 30\}$ | FFN 512–3840 | 128 | 3840 | 118.8–282.4 |
| Fully elastic | 64 | $\{6, 7, 8, 9, 10\}$ | 384–640 | 64 | 2560 | 119.0–287.5 |
| SHELLELASTIC | 128 | $\{4, 5, 6, 7, 8\}$ | 512–1024 | 128 | 1024 | 118.8–279.6 |
| *375M–906M scale* | | | | | | |
| Standalone | 64 | $\{M\}$, $M$ odd | 832–1344 | 64 | $2.75D_{\text{model}}$ | 377.5–901.6 |
| MatFormer | 128 | $\{8, 17, 27, 38, 51\}$ | FFN 1024–6528 | 128 | 6528 | 371.9–912.9 |
| Fully elastic | 64 | $\{13, 15, 17, 19, 21\}$ | 832–1344 | 64 | 3696 | 377.5–901.6 |
| SHELLELASTIC | 256 | $\{4, 5, 6, 7, 8\}$ | 1024–2048 | 128 | 2048 | 371.9–911.5 |
| *675M–1.69B scale* | | | | | | |
| Standalone | 64 | $\{M\}$, $M$ odd | 832–1344 | 64 | $7.25D_{\text{model}}$ | 676.6–1682.0 |
| MatFormer | 256 | $\{16, 24, 34, 45, 57\}$ | FFN 4096–14592 | 128 | 14592 | 673.8–1705.6 |
| Fully elastic | 64 | $\{13, 15, 17, 19, 21\}$ | 832–1344 | 64 | 9744 | 676.6–1682.0 |
| SHELLELASTIC | 256 | $\{4, 5, 6, 7, 8\}$ | 1024–2048 | 128 | 8192 | 673.8–1704.2 |

Table 7: **Architecture settings at the 119M–285M scale.** The upper block is shared by all models, and the lower block lists the settings that differ between them (base widths, supported sets and sizes of all three scales in Table 6).

| *Shared by all models* | |
|---|---|
| Layers | 32 pre-norm layers with residual-branch multiplier 1 |
| Normalization | RMSNorm with learnable gains and $\epsilon_N = 10^{-5}$; query–key RMSNorm per head |
| Attention | rotary embeddings with base 10,000, logit scale $d_{\text{head}}^{-1/2}$, elementwise sigmoid gate |
| FFN and biases | SwiGLU; no biases |
| Embeddings | untied, over the 50,432-token GPT-NeoX vocabulary |
| Initialization | truncated normal at $\pm 3$ standard deviations |

| Setting | Proxy | Standalone | MatFormer | Fully elastic | SHELLELASTIC |
|---|---|---|---|---|---|
| Elastic dimensions | – | – | FFN hidden width | residual, heads, FFN hidden width, embedding, readout | residual, heads, FFN hidden width, embedding, terminal norm, readout |
| Trunk normalization | RMSNorm | RMSNorm | RMSNorm | over the active width | group-local over $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ |
| Terminal norm and readout | single | single | shared, one learned $\alpha$ | shared and sliced, learned $\alpha_s$ | per width: $N_s^{\text{out}}$, $U_s$, learned $\alpha_s$ |
| Sub-networks per step | – | – | one, uniformly at random | one, uniformly at random | all five, $\omega_s = (1, 1, 1, 1, 4)$ |
| Parameterization | $\mu$P, $m = 1$ | $\mu$P, $m =$ width$/128$ | $\mu$-ELASTIC, $m = 4$ (Table 5) | $\mu$-ELASTIC, $m = 1/2$ (Table 4) | $\mu$-ELASTIC, $m = 1$ (Table 1) |
| Parameters (M) | 17.1 | 119.0, 154.4, 194.3, 238.7, 287.5 | 118.8, 150.3, 188.0, 232.1, 282.4 | 119.0, 154.4, 194.3, 238.7, 287.5 | 118.8, 152.7, 190.8, 233.1, 279.6 |

### F.4 TRAINING AND INFERENCE COST

**Compute.** By Equation 9, a step of SHELLELASTIC runs one trunk pass at the full width and adds only one terminal norm and readout per supported width. Updating every width once takes one trunk

Table 8: **Training HPs** used unless otherwise specified (Appendix F.1 lists the changes at the two larger scales). The base values $\eta_0$, $\sigma_0$ and $\lambda_0$ are scaled per parameter group by $\mu$-ELASTIC (Table 1 and Appendix E) or by $\mu$P; SP runs use them verbatim.

| HP | Value |
|---|---|
| Sequence length | 2,048 |
| Batch size | 128 sequences (262,144 tokens) |
| Training steps | 25,000 (6.55B tokens) |
| Learning-rate schedule | warmup-stable-decay: linear warmup for 1,000 steps, constant to step 20,000, linear decay to 0 at step 25,000 |
| Base learning rate $\eta_0$ | $2 \times 10^{-3}$ |
| Base initialization $\sigma_0$ | $3.3 \times 10^{-3}$, truncated normal at $\pm 3$ standard deviations |
| Optimizer | AdamW with decoupled weight decay |
| Base weight decay $\lambda_0$ | 0.01; none on norm gains and $\alpha_s$ |
| $\beta_1$, $\beta_2$ | 0.95, 0.95 |
| AdamW $\epsilon$ | $10^{-14}$ |
| Gradient clipping | global norm 4 |
| Step skipping | optimizer step skipped if the gradient norm exceeds $20\times$ the clip value, after step 5,000 |
| $z$-loss coefficient $\lambda_z$ | $10^{-4}$ |
| Precision | bf16 mixed precision with fp32 master weights and fp32 gradient reduction |
| Parallelism | hybrid-sharded data parallelism (4-GPU shards, two replicas) on one node of 8 A100, H100 or H200 GPUs |
| Seeds | 95 (model), 1 (data) |
| Validation | C4 validation loss every 2,500 steps, reported at step 25,000 |

Table 9: **Parameters (M) of the sub-networks of Table 2** at each scale and tier, counted as in Appendix F.3; Avg. averages the five tiers. Standalone, sampled standalone and the fully elastic model share the same five dense architectures.

| | Method | Low | Medium | High | Extra | Max | Avg. |
|---|---|---|---|---|---|---|---|
| 119M–285M | Standalone | 119.0 | 154.4 | 194.3 | 238.7 | 287.5 | 198.8 |
| | Sampled standalone | 119.0 | 154.4 | 194.3 | 238.7 | 287.5 | 198.8 |
| | MatFormer | 118.8 | 150.3 | 188.0 | 232.1 | 282.4 | 194.3 |
| | Fully elastic | 119.0 | 154.4 | 194.3 | 238.7 | 287.5 | 198.8 |
| | SHELLELASTIC | 118.8 | 152.7 | 190.8 | 233.1 | 279.6 | 195.0 |
| 375M–906M | Standalone | 377.5 | 487.7 | 611.8 | 749.8 | 901.6 | 625.7 |
| | Sampled standalone | 377.5 | 487.7 | 611.8 | 749.8 | 901.6 | 625.7 |
| | MatFormer | 371.9 | 485.1 | 610.9 | 749.3 | 912.9 | 626.0 |
| | Fully elastic | 377.5 | 487.7 | 611.8 | 749.8 | 901.6 | 625.7 |
| | SHELLELASTIC | 371.9 | 481.6 | 608.1 | 751.4 | 911.5 | 624.9 |
| 675M–1.69B | Standalone | 676.6 | 885.8 | 1123.2 | 1388.5 | 1682.0 | 1151.2 |
| | Sampled standalone | 676.6 | 885.8 | 1123.2 | 1388.5 | 1682.0 | 1151.2 |
| | MatFormer | 673.8 | 875.2 | 1126.8 | 1403.7 | 1705.6 | 1157.0 |
| | Fully elastic | 676.6 | 885.8 | 1123.2 | 1388.5 | 1682.0 | 1151.2 |
| | SHELLELASTIC | 673.8 | 878.0 | 1117.7 | 1393.1 | 1704.2 | 1153.4 |

pass per width with MatFormer, the fully elastic model and the standalone models, and sampling one width per step instead makes the cost vary from step to step.

**Measurement protocol.** For Table 10, each tier runs as its own sub-network on A100-40GB GPUs, with the model sharded as in training. Latency is the mean of three passes after two warm-up passes (forward and backward for training, forward only for inference), and peak training memory is PyTorch's peak allocated memory per GPU during a training step, which for SHELLELASTIC trains all five tiers.

Table 10: **Latency and peak training memory at the 119M–285M scale** on A100-40GB GPUs, for one GPU's batch of 16 sequences of 2,048 tokens, with each tier run as its own sub-network (protocol in Appendix F.4). Entries are means over runs, with the standard deviation as a subscript; Avg. averages the tiers. †Five separately trained models.

| Method | Tier 1 Low | Tier 2 Medium | Tier 3 High | Tier 4 Extra | Tier 5 Max | Avg. |
|---|---|---|---|---|---|---|
| *Train latency (ms, forward + backward)* | | | | | | |
| Standalone† | $570_{\pm 2}$ | $644_{\pm 3}$ | $723_{\pm 6}$ | $834_{\pm 5}$ | $917_{\pm 10}$ | $737_{\pm 3}$ |
| Sampled standalone | | | *as Standalone* | | | |
| MatFormer | $641_{\pm 1}$ | $677_{\pm 2}$ | $722_{\pm 4}$ | $776_{\pm 5}$ | $834_{\pm 6}$ | $730_{\pm 4}$ |
| Fully elastic | $597_{\pm 4}$ | $671_{\pm 3}$ | $748_{\pm 5}$ | $858_{\pm 7}$ | $933_{\pm 9}$ | $761_{\pm 6}$ |
| SHELLELASTIC | $437_{\pm 11}$ | $527_{\pm 13}$ | $603_{\pm 8}$ | $690_{\pm 2}$ | $747_{\pm 3}$ | $601_{\pm 6}$ |
| *Inference latency (ms, forward)* | | | | | | |
| Standalone† | $151_{\pm 2}$ | $175_{\pm 1}$ | $199_{\pm 2}$ | $228_{\pm 2}$ | $254_{\pm 2}$ | $201_{\pm 1}$ |
| Sampled standalone | | | *as Standalone* | | | |
| MatFormer | $175_{\pm 1}$ | $186_{\pm 1}$ | $200_{\pm 1}$ | $216_{\pm 2}$ | $235_{\pm 2}$ | $202_{\pm 1}$ |
| Fully elastic | $160_{\pm 2}$ | $183_{\pm 1}$ | $207_{\pm 1}$ | $236_{\pm 1}$ | $260_{\pm 1}$ | $209_{\pm 1}$ |
| SHELLELASTIC | $141_{\pm 10}$ | $167_{\pm 15}$ | $178_{\pm 12}$ | $205_{\pm 1}$ | $218_{\pm 2}$ | $182_{\pm 8}$ |
| *Peak training memory (GB per GPU)* | | | | | | |
| Standalone† | $15.0_{\pm .0}$ | $17.5_{\pm .0}$ | $20.1_{\pm .0}$ | $22.6_{\pm .0}$ | $25.2_{\pm .0}$ | $20.1_{\pm .0}$ |
| Sampled standalone | | | *as Standalone* | | | |
| MatFormer | $15.8_{\pm .0}$ | $17.9_{\pm .0}$ | $20.3_{\pm .0}$ | $23.2_{\pm .0}$ | $26.5_{\pm .0}$ | $20.7_{\pm .0}$ |
| Fully elastic | $16.0_{\pm .0}$ | $18.5_{\pm .0}$ | $20.9_{\pm .0}$ | $23.4_{\pm .0}$ | $25.8_{\pm .0}$ | $20.9_{\pm .0}$ |
| SHELLELASTIC | $21.6_{\pm .0}$ (a step that trains all five tiers) | | | | | – |

**Activation checkpointing.** All runs train with activation checkpointing on every other transformer block (16 of 32), whose forward pass is recomputed during the backward pass. Training latency therefore includes this recomputation (on H100 GPUs, the Standalone models train 14–19% faster without it), and peak training memory keeps only the inputs of the checkpointed blocks (without checkpointing, Standalone needs $1.9\times$ the memory); inference latency is a plain forward pass and is unaffected.

**Sampled standalone.** Sampled standalone trains the same five models as Standalone, one per step, so its per-tier latency and memory are those of Standalone: in its training runs, the logged latencies agree within 2%, and the 2.2–2.6 GB of extra memory it logged comes from our implementation keeping the four inactive models' weights and optimizer states on the GPU, which offloading them would remove.

**Observations.** Two observations emerge:

- **The shared trunk is the saving of the architecture.** One training step of SHELLELASTIC updates all five tiers, whereas updating every tier once takes five steps with MatFormer and the fully elastic model (one per tier) and one step of each model with Standalone.
- **Fused kernels lower the per-tier cost.** At equal parameters, all methods need about the same effective FLOPs per tier, so the lower per-tier latency and memory of SHELLELASTIC in Table 10 come from our fused implementation: Triton kernels compute the RMSNorms of every transformer block, including the query and key normalizations, and read the queries and keys directly from the output of the QKV projection without copying them.

## G EVALUATION DETAILS

**Validation loss.** The validation loss is the next-token cross-entropy on held-out C4 text, averaged over non-padded tokens, computed every 2,500 steps and reported at the final step. Each supported

Table 11: **Downstream evaluation suite.** All tasks are evaluated 5-shot; ARC-Easy, PIQA and HellaSwag are also probed during pretraining. MC: multiple-choice accuracy; EM: exact match.

| Field | Dataset | Split | #Items | Metric | Random |
|---|---|---|---|---|---|
| World knowledge | MMLU | test | 14,042 | MC accuracy | 25% |
| | ARC-Easy | test | 2,376 | MC accuracy | 25% |
| | ARC-Challenge | test | 1,172 | MC accuracy | 25% |
| Symbolic problem solving | SVAMP | test | 300 | EM | 0% |
| Commonsense reasoning | COPA | val | 100 | MC accuracy | 50% |
| | PIQA | val | 1,838 | MC accuracy | 50% |
| Language understanding | HellaSwag | val | 10,042 | MC accuracy | 25% |
| | WinoGrande | val | 1,267 | cloze accuracy | 50% |
| | LAMBADA | test | 5,153 | final-word EM | 0% |
| Reading comprehension | BoolQ | val | 3,270 | MC accuracy | 50% |
| | AGIEval LSAT-RC | test | 268 | MC accuracy | 25% |
| | AGIEval LSAT-LR | test | 510 | MC accuracy | 25% |
| | AGIEval SAT-EN | test | 206 | MC accuracy | 25% |

sub-network is extracted from the supernet as in Section 3.3 and evaluated on its own, on the same validation batches as every other model, so losses are comparable across widths, families and the standalone models.

**Downstream tasks.** During pretraining, we also probe 5-shot accuracy on ARC-Easy, PIQA and HellaSwag at the same interval. After pretraining, we evaluate every sub-network on 13 datasets spanning five skill areas, following the protocol of DCLM and OLMoE, in a 5-shot setting on the standard community splits (Table 11). Multiple-choice tasks are scored by accuracy, with a random baseline of 25% for four-choice and 50% for two-choice items; WinoGrande uses its cloze formulation, LAMBADA is scored by exact match of the final word, and SVAMP by exact match of the generated numerical answer. For elastic models, we report the accuracy of every sub-network and their mean. The datasets of each skill area are described below.

**World knowledge.** Exam-style and science questions that probe academic knowledge.

- **MMLU** (Hendrycks et al., 2020) (14,042 items, four-choice): exam-style questions from 57 subjects across the sciences, humanities and social sciences.
- **ARC-Easy** and **ARC-Challenge** (Clark et al., 2018) (2,376 and 1,172 items, four-choice): grade-school science questions; the Challenge subset holds the questions that retrieval and co-occurrence methods answer incorrectly.

**Symbolic problem solving.** Arithmetic reasoning with a generated answer.

- **SVAMP** (Patel et al., 2021) (300 problems, free-response): short arithmetic word problems with numerical answers; the model generates a derivation followed by the answer, which is scored by exact match.

**Commonsense reasoning.** Everyday causal and physical knowledge.

- **COPA** (Gordon et al., 2012) (100 items, two-choice): choose the more plausible cause or effect of a premise.
- **PIQA** (Bisk et al., 2020) (1,838 items, two-choice): choose the more sensible solution to a physical task.

**Language understanding.** Narrative coherence, coreference and the use of broad context.

- **HellaSwag** (Zellers et al., 2019) (10,042 items, four-choice): choose the most plausible continuation of a partial narrative.

- **WinoGrande** (Sakaguchi et al., 2021) (1,267 items, two-choice): fill a blank with one of two candidates so that the sentence is coherent.
- **LAMBADA** (Paperno et al., 2016) (5,153 passages): predict the final word of a passage that requires broad discourse context.

**Reading comprehension.** Questions answered from a given passage or argument.

- **BoolQ** (Clark et al., 2019) (3,270 items, yes/no): answer a binary question about a short passage.
- **AGIEval LSAT-RC, LSAT-LR and SAT-EN** (Zhong et al., 2024) (268, 510 and 206 items, four-choice): LSAT reading-comprehension and logical-reasoning questions and SAT English questions.

## H COMPLETE MAIN RESULTS

Tables 12–26 give every metric of every sub-network in Table 2, one table per scale and tier, and Figures 7–9 show the training curves of each scale:

- **119M–285M scale** (25,000 steps, three runs): Figure 7 and Tables 12–16;
- **375M–906M scale** (50,000 steps, single runs): Figure 8 and Tables 17–21;
- **675M–1.69B scale** (100,000 steps, single runs): Figure 9 and Tables 22–26.

**Tables.** Each table gives the parameter count, the train loss (cross-entropy without the $z$-loss, mean over the last 500 steps), the validation loss at the last step, and the 5-shot accuracy (%) on each of the 13 downstream tasks of Appendix G, grouped by field as there (LSAT-RC, LSAT-LR and SAT-EN are from AGIEval), with their average (the accuracy of Table 2). At the 119M–285M scale, entries are the mean over three runs of the same configuration and seed, with the standard deviation as a subscript; at the two larger scales, every entry is a single run. Standalone$^{\dagger}$ is five separately trained models, each trained for the full number of steps, which together take $5\times$ the optimizer steps of any other column. Bold marks the best entry among the four methods that train all five sizes within one run.

**Curves.** Each figure has one column per tier and plots, against the global training step, the train loss (the logged means over 50-step intervals), the validation loss, and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA, the three benchmarks evaluated during training. Sampled standalone, MatFormer and the fully elastic model sample one sub-network uniformly at random per step, SHELLELASTIC updates all sizes at every step, and Standalone trains each size separately for the full number of steps ($5\times$ the steps). Values are plotted as logged, without smoothing or clipping, and y-axes are shared within each row.

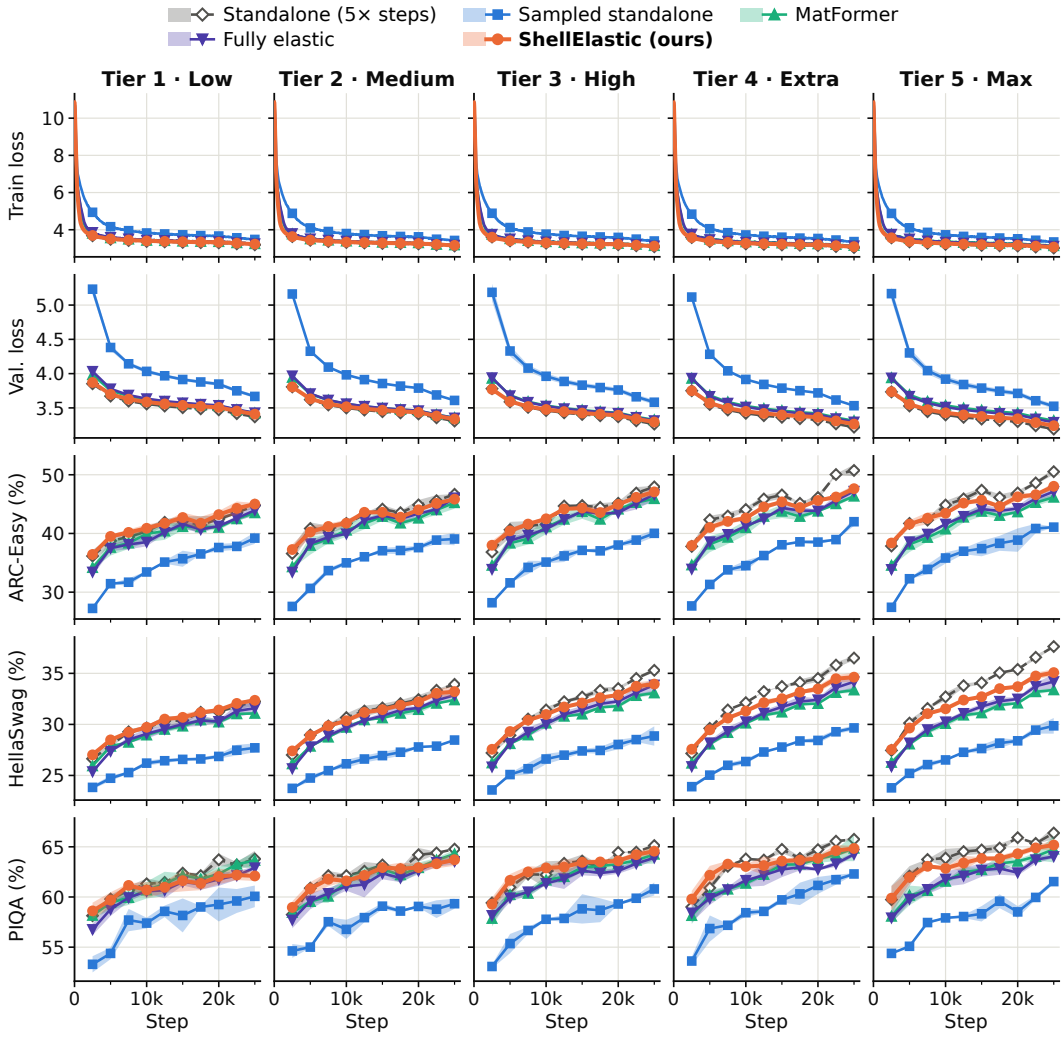


Figure 7: **Training curves at the 119M–285M scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 119M parameters) to Tier 5 (Max, about 285M); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Lines are means over three runs, with shaded bands of $\pm 1$ standard deviation; conventions in Appendix H.

Table 12: **Per-task results at the 119M–285M scale, Tier 1 (Low, about 119M parameters).** Sub-network width 384 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 512 at $D_{\text{model}} = 512$ for MatFormer, and 512 for SHELLELASTIC. Mean over three runs, with the standard deviation as a subscript.

| **Tier 1 · Low** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 119.0 | 119.0 | 118.8 | 119.0 | 118.8 |
| Train loss ↓ | $3.1951_{\pm.0035}$ | $3.4911_{\pm.0235}$ | $3.2536_{\pm.0032}$ | $3.2572_{\pm.0091}$ | $\mathbf{3.2305}_{\pm.0015}$ |
| Val. loss ↓ | $3.3695_{\pm.0034}$ | $3.6674_{\pm.0242}$ | $3.4248_{\pm.0036}$ | $3.4291_{\pm.0097}$ | $\mathbf{3.4086}_{\pm.0009}$ |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | $24.5_{\pm.3}$ | $24.4_{\pm.2}$ | $\mathbf{24.8}_{\pm.5}$ | $24.7_{\pm.7}$ | $24.7_{\pm.6}$ |
| ARC-Easy | $44.8_{\pm.5}$ | $39.2_{\pm1.0}$ | $43.5_{\pm.5}$ | $44.0_{\pm.9}$ | $\mathbf{45.0}_{\pm.2}$ |
| ARC-Challenge | $23.3_{\pm1.7}$ | $22.7_{\pm.5}$ | $\mathbf{23.9}_{\pm.7}$ | $23.3_{\pm1.2}$ | $23.3_{\pm.5}$ |
| SVAMP | $5.7_{\pm.5}$ | $6.3_{\pm.9}$ | $\mathbf{6.8}_{\pm1.5}$ | $6.4_{\pm.8}$ | $5.7_{\pm.3}$ |
| COPA | $61.0_{\pm.0}$ | $54.0_{\pm5.7}$ | $57.7_{\pm.6}$ | $58.7_{\pm1.5}$ | $\mathbf{59.7}_{\pm.6}$ |
| PIQA | $63.8_{\pm.8}$ | $60.1_{\pm1.1}$ | $\mathbf{63.6}_{\pm.8}$ | $62.9_{\pm.2}$ | $62.1_{\pm.5}$ |
| HellaSwag | $32.2_{\pm.4}$ | $27.7_{\pm.4}$ | $31.1_{\pm.1}$ | $31.6_{\pm.5}$ | $\mathbf{32.4}_{\pm.2}$ |
| WinoGrande | $52.5_{\pm.1}$ | $\mathbf{52.1}_{\pm.2}$ | $50.7_{\pm.8}$ | $49.0_{\pm1.3}$ | $50.7_{\pm.8}$ |
| LAMBADA | $25.7_{\pm1.1}$ | $17.4_{\pm1.7}$ | $24.7_{\pm.4}$ | $23.6_{\pm1.0}$ | $\mathbf{25.9}_{\pm.7}$ |
| BoolQ | $51.1_{\pm7.2}$ | $54.9_{\pm7.6}$ | $50.4_{\pm5.8}$ | $53.0_{\pm.6}$ | $\mathbf{55.1}_{\pm7.2}$ |
| LSAT-RC | $25.2_{\pm1.3}$ | $\mathbf{25.2}_{\pm.3}$ | $24.3_{\pm1.7}$ | $25.1_{\pm1.8}$ | $24.8_{\pm1.8}$ |
| LSAT-LR | $26.9_{\pm.0}$ | $\mathbf{28.8}_{\pm.3}$ | $27.9_{\pm.6}$ | $25.4_{\pm1.3}$ | $26.2_{\pm1.0}$ |
| SAT-EN | $26.5_{\pm1.7}$ | $\mathbf{26.9}_{\pm.3}$ | $26.7_{\pm1.3}$ | $26.1_{\pm1.7}$ | $25.6_{\pm.3}$ |
| Average | $35.6_{\pm.9}$ | $33.8_{\pm.2}$ | $35.1_{\pm.3}$ | $34.9_{\pm.2}$ | $\mathbf{35.5}_{\pm.4}$ |

Table 13: **Per-task results at the 119M–285M scale, Tier 2 (Medium, about 153M parameters).** Sub-network width 448 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1152 at $D_{\text{model}} = 512$ for MatFormer, and 640 for SHELLELASTIC. Mean over three runs, with the standard deviation as a subscript.

| **Tier 2 · Medium** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 154.4 | 154.4 | 150.3 | 154.4 | 152.7 |
| Train loss ↓ | $3.1344_{\pm.0019}$ | $3.4301_{\pm.0095}$ | $3.1821_{\pm.0034}$ | $3.1783_{\pm.0083}$ | $\mathbf{3.1641}_{\pm.0031}$ |
| Val. loss ↓ | $3.3081_{\pm.0016}$ | $3.6091_{\pm.0104}$ | $3.3566_{\pm.0037}$ | $3.3536_{\pm.0083}$ | $\mathbf{3.3375}_{\pm.0027}$ |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | $25.0_{\pm.4}$ | $\mathbf{24.9}_{\pm.3}$ | $\mathbf{24.9}_{\pm.2}$ | $24.7_{\pm.3}$ | $23.8_{\pm.7}$ |
| ARC-Easy | $46.7_{\pm.1}$ | $39.1_{\pm1.1}$ | $45.2_{\pm1.0}$ | $\mathbf{46.1}_{\pm.8}$ | $45.8_{\pm.1}$ |
| ARC-Challenge | $24.4_{\pm.4}$ | $23.0_{\pm.7}$ | $23.6_{\pm.0}$ | $\mathbf{24.0}_{\pm.4}$ | $23.5_{\pm.4}$ |
| SVAMP | $6.8_{\pm.2}$ | $5.8_{\pm.7}$ | $6.7_{\pm1.5}$ | $\mathbf{7.2}_{\pm1.4}$ | $6.3_{\pm.9}$ |
| COPA | $59.0_{\pm1.4}$ | $58.0_{\pm1.4}$ | $60.0_{\pm1.7}$ | $60.3_{\pm1.5}$ | $\mathbf{61.7}_{\pm1.2}$ |
| PIQA | $64.8_{\pm.0}$ | $59.3_{\pm.5}$ | $\mathbf{64.3}_{\pm.4}$ | $63.5_{\pm.3}$ | $63.7_{\pm.6}$ |
| HellaSwag | $33.9_{\pm.1}$ | $28.5_{\pm.1}$ | $32.4_{\pm.1}$ | $32.9_{\pm.3}$ | $\mathbf{33.2}_{\pm.5}$ |
| WinoGrande | $51.7_{\pm2.2}$ | $\mathbf{51.5}_{\pm2.7}$ | $50.5_{\pm.6}$ | $50.1_{\pm.3}$ | $49.3_{\pm1.8}$ |
| LAMBADA | $28.4_{\pm.3}$ | $18.9_{\pm.3}$ | $26.4_{\pm.6}$ | $26.4_{\pm1.2}$ | $\mathbf{29.1}_{\pm.7}$ |
| BoolQ | $49.9_{\pm4.8}$ | $48.1_{\pm6.2}$ | $50.3_{\pm6.1}$ | $50.7_{\pm.6}$ | $\mathbf{57.2}_{\pm3.4}$ |
| LSAT-RC | $24.6_{\pm4.2}$ | $24.8_{\pm2.9}$ | $22.0_{\pm3.3}$ | $24.6_{\pm.4}$ | $\mathbf{25.6}_{\pm1.4}$ |
| LSAT-LR | $24.9_{\pm.6}$ | $26.3_{\pm1.9}$ | $25.9_{\pm1.5}$ | $26.3_{\pm3.0}$ | $\mathbf{29.2}_{\pm.9}$ |
| SAT-EN | $25.7_{\pm.0}$ | $25.5_{\pm1.7}$ | $25.2_{\pm3.0}$ | $24.8_{\pm1.3}$ | $\mathbf{27.2}_{\pm3.0}$ |
| Average | $35.8_{\pm1.0}$ | $33.4_{\pm1.1}$ | $35.2_{\pm.2}$ | $35.5_{\pm.4}$ | $\mathbf{36.6}_{\pm.6}$ |

Table 14: **Per-task results at the 119M–285M scale, Tier 3 (High, about 192M parameters).** Sub-network width 512 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1920 at $D_{\text{model}} = 512$ for MatFormer, and 768 for SHELLELASTIC. Mean over three runs, with the standard deviation as a subscript.

| **Tier 3 · High** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 194.3 | 194.3 | 188.0 | 194.3 | 190.8 |
| Train loss ↓ | $3.0916_{\pm.0086}$ | $3.4087_{\pm.0336}$ | $3.1561_{\pm.0036}$ | $3.1447_{\pm.0077}$ | $\mathbf{3.1229}_{\pm.0018}$ |
| Val. loss ↓ | $3.2635_{\pm.0092}$ | $3.5826_{\pm.0344}$ | $3.3250_{\pm.0036}$ | $3.3145_{\pm.0076}$ | $\mathbf{3.2944}_{\pm.0016}$ |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | $24.6_{\pm.4}$ | $\mathbf{25.0}_{\pm.3}$ | $24.6_{\pm.4}$ | $24.6_{\pm.5}$ | $24.0_{\pm.5}$ |
| ARC-Easy | $47.9_{\pm.2}$ | $40.0_{\pm.6}$ | $45.9_{\pm.7}$ | $46.6_{\pm1.0}$ | $\mathbf{47.1}_{\pm.8}$ |
| ARC-Challenge | $23.9_{\pm.8}$ | $22.3_{\pm.8}$ | $24.3_{\pm.3}$ | $\mathbf{24.5}_{\pm.7}$ | $23.6_{\pm.8}$ |
| SVAMP | $7.7_{\pm1.9}$ | $6.3_{\pm.0}$ | $6.6_{\pm1.5}$ | $\mathbf{6.8}_{\pm3.3}$ | $6.3_{\pm1.0}$ |
| COPA | $60.5_{\pm2.1}$ | $55.0_{\pm1.4}$ | $60.3_{\pm2.3}$ | $60.3_{\pm3.1}$ | $\mathbf{61.7}_{\pm3.1}$ |
| PIQA | $65.1_{\pm.1}$ | $60.8_{\pm.7}$ | $64.3_{\pm.9}$ | $63.9_{\pm.3}$ | $\mathbf{64.6}_{\pm.4}$ |
| HellaSwag | $35.3_{\pm.2}$ | $28.9_{\pm1.0}$ | $33.1_{\pm.2}$ | $\mathbf{33.9}_{\pm.6}$ | $\mathbf{33.9}_{\pm.3}$ |
| WinoGrande | $52.0_{\pm.6}$ | $\mathbf{50.9}_{\pm.6}$ | $50.5_{\pm.8}$ | $50.2_{\pm.5}$ | $50.0_{\pm2.1}$ |
| LAMBADA | $29.6_{\pm.1}$ | $19.3_{\pm1.0}$ | $27.6_{\pm.4}$ | $28.0_{\pm1.4}$ | $\mathbf{29.9}_{\pm1.1}$ |
| BoolQ | $54.1_{\pm.1}$ | $53.9_{\pm.8}$ | $51.1_{\pm7.6}$ | $50.1_{\pm3.5}$ | $\mathbf{55.3}_{\pm1.4}$ |
| LSAT-RC | $25.6_{\pm.3}$ | $\mathbf{26.3}_{\pm2.4}$ | $23.6_{\pm1.3}$ | $24.0_{\pm1.6}$ | $26.0_{\pm1.5}$ |
| LSAT-LR | $24.9_{\pm.6}$ | $\mathbf{27.1}_{\pm1.7}$ | $\mathbf{27.1}_{\pm1.2}$ | $25.7_{\pm2.2}$ | $26.8_{\pm1.6}$ |
| SAT-EN | $24.8_{\pm.7}$ | $\mathbf{26.5}_{\pm1.0}$ | $\mathbf{26.5}_{\pm3.0}$ | $25.2_{\pm1.5}$ | $26.2_{\pm1.7}$ |
| Average | $36.6_{\pm.2}$ | $34.0_{\pm.7}$ | $35.8_{\pm.5}$ | $35.7_{\pm.4}$ | $\mathbf{36.6}_{\pm.4}$ |

Table 15: **Per-task results at the 119M–285M scale, Tier 4 (Extra, about 236M parameters).** Sub-network width 576 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 2816 at $D_{\text{model}} = 512$ for MatFormer, and 896 for SHELLELASTIC. Mean over three runs, with the standard deviation as a subscript.

| **Tier 4 · Extra** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 238.7 | 238.7 | 232.1 | 238.7 | 233.1 |
| Train loss ↓ | $3.0474_{\pm.0006}$ | $3.3561_{\pm.0056}$ | $3.1407_{\pm.0036}$ | $3.1233_{\pm.0071}$ | $\mathbf{3.0941}_{\pm.0015}$ |
| Val. loss ↓ | $3.2186_{\pm.0012}$ | $3.5319_{\pm.0056}$ | $3.3123_{\pm.0037}$ | $3.2954_{\pm.0072}$ | $\mathbf{3.2641}_{\pm.0017}$ |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | $24.6_{\pm.5}$ | $\mathbf{24.7}_{\pm.1}$ | $24.5_{\pm.4}$ | $24.6_{\pm.5}$ | $24.4_{\pm.7}$ |
| ARC-Easy | $50.7_{\pm1.0}$ | $42.0_{\pm.7}$ | $46.3_{\pm.2}$ | $47.2_{\pm.9}$ | $\mathbf{47.6}_{\pm.6}$ |
| ARC-Challenge | $25.2_{\pm1.1}$ | $23.2_{\pm.4}$ | $24.0_{\pm.5}$ | $\mathbf{24.8}_{\pm1.1}$ | $24.0_{\pm.5}$ |
| SVAMP | $8.0_{\pm.9}$ | $6.5_{\pm.7}$ | $\mathbf{7.0}_{\pm.6}$ | $6.2_{\pm2.2}$ | $6.7_{\pm1.5}$ |
| COPA | $61.0_{\pm2.8}$ | $57.5_{\pm.7}$ | $60.7_{\pm.6}$ | $61.0_{\pm.0}$ | $\mathbf{63.0}_{\pm2.6}$ |
| PIQA | $65.8_{\pm.2}$ | $62.3_{\pm.1}$ | $\mathbf{64.9}_{\pm1.3}$ | $64.3_{\pm.2}$ | $64.8_{\pm.3}$ |
| HellaSwag | $36.5_{\pm.2}$ | $29.7_{\pm.2}$ | $33.4_{\pm.3}$ | $34.2_{\pm.6}$ | $\mathbf{34.6}_{\pm.4}$ |
| WinoGrande | $50.7_{\pm.1}$ | $48.6_{\pm.4}$ | $50.1_{\pm.3}$ | $\mathbf{50.2}_{\pm.6}$ | $49.8_{\pm1.4}$ |
| LAMBADA | $31.7_{\pm.9}$ | $20.4_{\pm.4}$ | $28.0_{\pm.4}$ | $29.2_{\pm.9}$ | $\mathbf{30.2}_{\pm.9}$ |
| BoolQ | $50.7_{\pm1.6}$ | $51.3_{\pm5.0}$ | $50.4_{\pm7.7}$ | $49.2_{\pm2.7}$ | $\mathbf{54.5}_{\pm2.4}$ |
| LSAT-RC | $25.7_{\pm1.6}$ | $25.2_{\pm.3}$ | $23.4_{\pm.9}$ | $24.1_{\pm1.5}$ | $\mathbf{26.4}_{\pm1.1}$ |
| LSAT-LR | $26.5_{\pm3.6}$ | $27.8_{\pm1.1}$ | $\mathbf{28.0}_{\pm2.1}$ | $25.2_{\pm1.4}$ | $27.6_{\pm2.2}$ |
| SAT-EN | $24.8_{\pm3.4}$ | $25.0_{\pm.3}$ | $26.2_{\pm2.6}$ | $\mathbf{26.5}_{\pm.7}$ | $\mathbf{26.5}_{\pm1.2}$ |
| Average | $37.1_{\pm.3}$ | $34.2_{\pm.1}$ | $35.9_{\pm.6}$ | $35.9_{\pm.3}$ | $\mathbf{36.9}_{\pm.2}$ |

Table 16: **Per-task results at the 119M–285M scale, Tier 5 (Max, about 285M parameters).** Sub-network width 640 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 3840 at $D_{\text{model}} = 512$ for MatFormer, and 1024 for SHELLELASTIC. Mean over three runs, with the standard deviation as a subscript.

| **Tier 5 · Max** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 287.5 | 287.5 | 282.4 | 287.5 | 279.6 |
| Train loss ↓ | $3.0203_{\pm.0017}$ | $3.3505_{\pm.0321}$ | $3.1436_{\pm.0035}$ | $3.1196_{\pm.0070}$ | $\mathbf{3.0713}_{\pm.0014}$ |
| Val. loss ↓ | $3.1908_{\pm.0014}$ | $3.5239_{\pm.0340}$ | $3.3139_{\pm.0036}$ | $3.2899_{\pm.0071}$ | $\mathbf{3.2403}_{\pm.0016}$ |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | $24.8_{\pm.9}$ | $24.5_{\pm.2}$ | $\mathbf{24.8}_{\pm.4}$ | $24.3_{\pm.2}$ | $24.2_{\pm.4}$ |
| ARC-Easy | $50.5_{\pm.5}$ | $41.1_{\pm.1}$ | $46.2_{\pm.5}$ | $47.1_{\pm.9}$ | $\mathbf{48.0}_{\pm.7}$ |
| ARC-Challenge | $25.4_{\pm.4}$ | $23.2_{\pm1.1}$ | $24.2_{\pm.4}$ | $\mathbf{25.1}_{\pm1.2}$ | $24.3_{\pm.8}$ |
| SVAMP | $5.8_{\pm.7}$ | $6.3_{\pm1.4}$ | $\mathbf{7.2}_{\pm1.0}$ | $5.7_{\pm2.4}$ | $6.4_{\pm1.2}$ |
| COPA | $61.0_{\pm.0}$ | $58.0_{\pm2.8}$ | $60.0_{\pm1.0}$ | $60.7_{\pm1.2}$ | $\mathbf{63.3}_{\pm2.9}$ |
| PIQA | $66.4_{\pm.6}$ | $61.5_{\pm.2}$ | $64.8_{\pm1.3}$ | $64.0_{\pm.1}$ | $\mathbf{65.2}_{\pm.5}$ |
| HellaSwag | $37.6_{\pm.1}$ | $29.9_{\pm.8}$ | $33.4_{\pm.1}$ | $34.2_{\pm.7}$ | $\mathbf{35.1}_{\pm.3}$ |
| WinoGrande | $50.7_{\pm.1}$ | $49.8_{\pm1.7}$ | $\mathbf{50.4}_{\pm.9}$ | $49.9_{\pm1.1}$ | $49.8_{\pm1.4}$ |
| LAMBADA | $33.0_{\pm1.9}$ | $20.0_{\pm.4}$ | $27.8_{\pm.4}$ | $29.3_{\pm1.3}$ | $\mathbf{30.4}_{\pm.4}$ |
| BoolQ | $49.7_{\pm4.2}$ | $\mathbf{55.8}_{\pm5.4}$ | $50.7_{\pm7.3}$ | $50.6_{\pm3.4}$ | $52.3_{\pm3.1}$ |
| LSAT-RC | $25.6_{\pm.3}$ | $23.9_{\pm.5}$ | $25.1_{\pm.6}$ | $25.0_{\pm.6}$ | $\mathbf{25.6}_{\pm1.1}$ |
| LSAT-LR | $25.7_{\pm2.2}$ | $27.2_{\pm.1}$ | $27.0_{\pm3.0}$ | $25.4_{\pm2.3}$ | $\mathbf{27.6}_{\pm.6}$ |
| SAT-EN | $26.2_{\pm1.4}$ | $26.5_{\pm2.4}$ | $25.2_{\pm1.0}$ | $\mathbf{26.9}_{\pm.7}$ | $\mathbf{26.9}_{\pm2.0}$ |
| Average | $37.1_{\pm.7}$ | $34.4_{\pm.5}$ | $35.9_{\pm.7}$ | $36.0_{\pm.5}$ | $\mathbf{36.9}_{\pm.5}$ |

Table 17: **Per-task results at the 375M–906M scale, Tier 1 (Low, about 375M parameters).** Sub-network width 832 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 1024 at $D_{\text{model}} = 1024$ for MatFormer, and 1024 for SHELLELASTIC. Single runs.

| **Tier 1 · Low** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 377.5 | 377.5 | 371.9 | 377.5 | 371.9 |
| Train loss ↓ | 2.8292 | 3.0370 | 2.9009 | 2.8796 | **2.8724** |
| Val. loss ↓ | 3.0062 | 3.2069 | 3.0741 | 3.0531 | **3.0528** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 24.4 | **25.3** | 24.8 | 23.2 | 24.8 |
| ARC-Easy | 58.2 | 50.6 | 55.4 | **56.0** | 55.0 |
| ARC-Challenge | 29.4 | 26.2 | **28.8** | 27.4 | 27.8 |
| SVAMP | 7.7 | **7.3** | 6.0 | **7.3** | 6.3 |
| COPA | 67.0 | 58.0 | **69.0** | 66.0 | 67.0 |
| PIQA | 69.7 | 66.8 | 68.8 | 68.5 | **69.7** |
| HellaSwag | 46.7 | 37.3 | 43.0 | 44.6 | **45.8** |
| WinoGrande | 56.0 | 51.6 | 54.0 | 53.3 | **54.5** |
| LAMBADA | 43.9 | 33.3 | 40.7 | 41.3 | **44.8** |
| BoolQ | 49.8 | 54.2 | 57.2 | **59.1** | 58.8 |
| LSAT-RC | 23.1 | 24.3 | **26.5** | 18.3 | 25.0 |
| LSAT-LR | 23.7 | **27.1** | 26.7 | 26.5 | 26.5 |
| SAT-EN | 24.8 | 25.7 | **29.1** | 28.2 | 23.8 |
| Average | 40.3 | 37.5 | **40.8** | 40.0 | **40.8** |

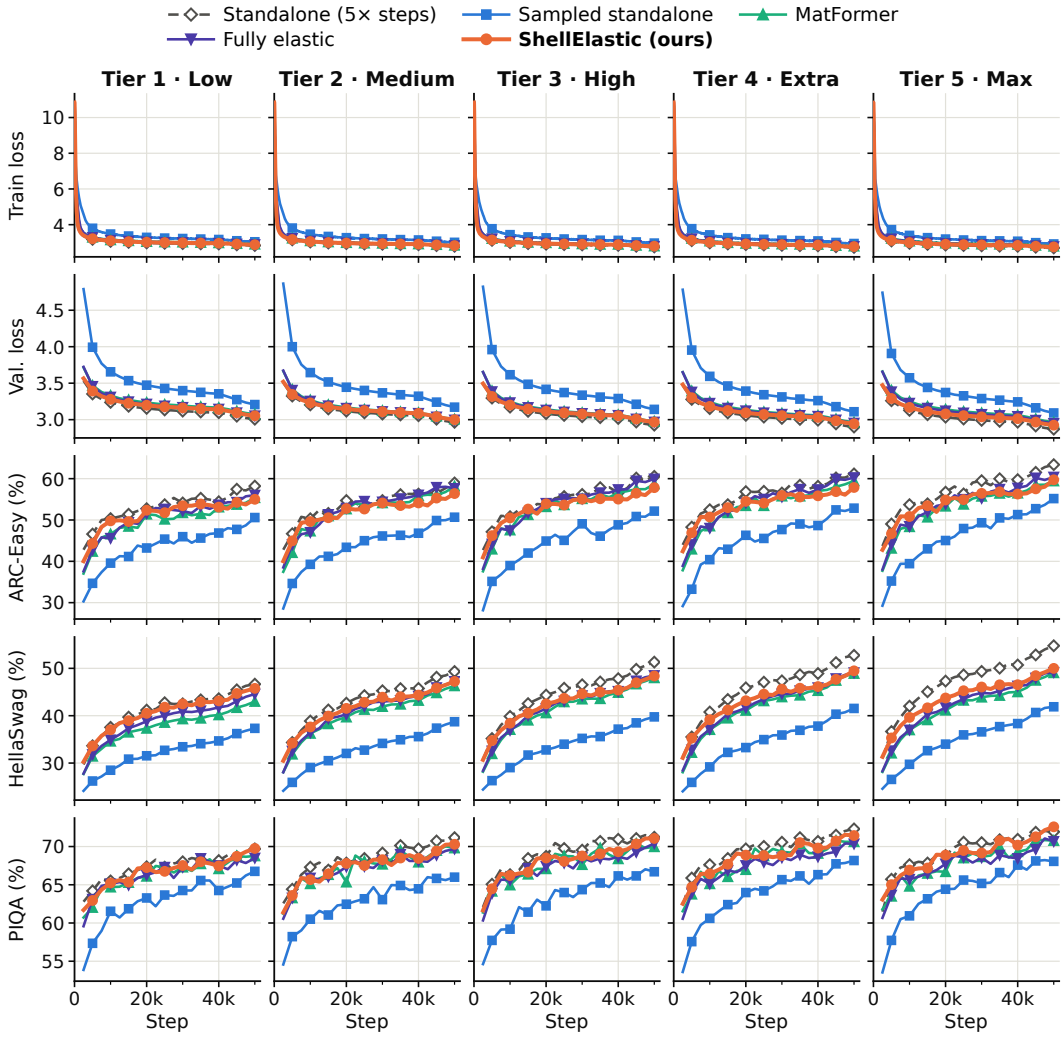


Figure 8: **Training curves at the 375M–906M scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 375M parameters) to Tier 5 (Max, about 906M); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Each line is a single run; conventions in Appendix H.

Table 18: **Per-task results at the 375M–906M scale, Tier 2 (Medium, about 486M parameters).** Sub-network width 960 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 2176 at $D_{\text{model}} = 1024$ for MatFormer, and 1280 for SHELLELASTIC. Single runs.

| **Tier 2 · Medium** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 487.7 | 487.7 | 485.1 | 487.7 | 481.6 |
| Train loss ↓ | 2.7866 | 2.9969 | 2.8300 | **2.8183** | 2.8219 |
| Val. loss ↓ | 2.9636 | 3.1713 | 3.0066 | **2.9963** | 2.9998 |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 25.4 | 24.4 | 24.8 | **25.3** | 24.4 |
| ARC-Easy | 59.0 | 50.7 | **58.2** | 57.7 | 56.4 |
| ARC-Challenge | 30.5 | 25.8 | **29.4** | 29.1 | 28.6 |
| SVAMP | 6.3 | 6.3 | **7.7** | 6.0 | 6.7 |
| COPA | 67.0 | 62.0 | 65.0 | **69.0** | 67.0 |
| PIQA | 71.2 | 66.0 | 69.8 | 69.7 | **70.3** |
| HellaSwag | 49.3 | 38.7 | 46.3 | **47.2** | **47.2** |
| WinoGrande | 55.1 | 52.2 | **54.1** | **54.1** | 53.8 |
| LAMBADA | 44.9 | 34.7 | 44.2 | 44.2 | **45.6** |
| BoolQ | 54.3 | 51.9 | **58.5** | 55.7 | 57.7 |
| LSAT-RC | 26.5 | 25.4 | 22.4 | 23.9 | **29.1** |
| LSAT-LR | 25.1 | 25.7 | 22.0 | 24.9 | **29.6** |
| SAT-EN | 23.8 | 26.2 | 24.8 | 26.2 | **26.7** |
| Average | 41.4 | 37.7 | 40.5 | 41.0 | **41.8** |

Table 19: **Per-task results at the 375M–906M scale, Tier 3 (High, about 611M parameters).** Sub-network width 1088 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 3456 at $D_{\text{model}} = 1024$ for MatFormer, and 1536 for SHELLELASTIC. Single runs.

| **Tier 3 · High** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 611.8 | 611.8 | 610.9 | 611.8 | 608.1 |
| Train loss ↓ | 2.7479 | 2.9720 | 2.8053 | 2.7942 | **2.7897** |
| Val. loss ↓ | 2.9245 | 3.1408 | 2.9768 | 2.9677 | **2.9670** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 23.7 | 23.6 | 25.2 | **25.7** | 24.5 |
| ARC-Easy | 60.6 | 52.1 | 58.7 | **59.9** | 57.8 |
| ARC-Challenge | 31.3 | 26.9 | 29.7 | **29.9** | 29.3 |
| SVAMP | 4.7 | 6.3 | 5.7 | 5.0 | **8.0** |
| COPA | 68.0 | 66.0 | 66.0 | **69.0** | 68.0 |
| PIQA | 71.2 | 66.7 | 70.0 | 70.4 | **71.1** |
| HellaSwag | 51.3 | 39.8 | 48.0 | **48.5** | 48.4 |
| WinoGrande | 54.9 | 52.1 | 52.8 | 53.4 | **54.4** |
| LAMBADA | 48.4 | 34.8 | **45.5** | **45.5** | **45.5** |
| BoolQ | 51.7 | 51.8 | 57.6 | 59.3 | **60.0** |
| LSAT-RC | 29.1 | 22.4 | 22.0 | **27.6** | 27.2 |
| LSAT-LR | 24.3 | 26.1 | 24.9 | 25.3 | **28.8** |
| SAT-EN | 30.6 | 24.3 | 21.4 | 27.2 | **28.6** |
| Average | 42.3 | 37.9 | 40.6 | 42.1 | **42.4** |

Table 20: **Per-task results at the 375M–906M scale, Tier 4 (Extra, about 750M parameters).** Sub-network width 1216 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 4864 at $D_{\text{model}} = 1024$ for MatFormer, and 1792 for SHELLELASTIC. Single runs.

| **Tier 4 · Extra** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 749.8 | 749.8 | 749.3 | 749.8 | 751.4 |
| Train loss ↓ | 2.7216 | 2.9340 | 2.7874 | 2.7742 | **2.7667** |
| Val. loss ↓ | 2.8987 | 3.1084 | 2.9647 | 2.9533 | **2.9432** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 25.3 | 23.8 | 24.6 | **25.0** | 24.2 |
| ARC-Easy | 61.2 | 52.9 | 59.5 | **60.3** | 57.9 |
| ARC-Challenge | 31.9 | 26.3 | **30.9** | 29.8 | 29.7 |
| SVAMP | 5.7 | 5.7 | 5.0 | 5.3 | **6.7** |
| COPA | 68.0 | 69.0 | 66.0 | 69.0 | **70.0** |
| PIQA | 72.3 | 68.2 | 70.6 | 70.3 | **71.4** |
| HellaSwag | 52.7 | 41.5 | 48.9 | 49.1 | **49.5** |
| WinoGrande | 55.6 | 53.6 | 54.1 | 53.4 | **54.7** |
| LAMBADA | 49.3 | 39.7 | **46.2** | 45.7 | 45.7 |
| BoolQ | 58.6 | 56.0 | 59.0 | **59.8** | 57.6 |
| LSAT-RC | 24.6 | 21.3 | 23.9 | 26.1 | **28.7** |
| LSAT-LR | 23.5 | 24.5 | 24.9 | 24.1 | **30.0** |
| SAT-EN | 27.7 | 24.8 | 26.2 | **29.1** | 28.6 |
| Average | 42.8 | 39.0 | 41.5 | 42.1 | **42.7** |

Table 21: **Per-task results at the 375M–906M scale, Tier 5 (Max, about 906M parameters).** Sub-network width 1344 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 6528 at $D_{\text{model}} = 1024$ for MatFormer, and 2048 for SHELLELASTIC. Single runs.

| **Tier 5 · Max** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 901.6 | 901.6 | 912.9 | 901.6 | 911.5 |
| Train loss ↓ | 2.6922 | 2.9186 | 2.7888 | 2.7716 | **2.7482** |
| Val. loss ↓ | 2.8693 | 3.0918 | 2.9650 | 2.9495 | **2.9245** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 24.7 | 24.6 | 24.8 | **25.4** | 24.7 |
| ARC-Easy | 63.4 | 55.2 | 59.4 | **60.4** | 59.7 |
| ARC-Challenge | 32.4 | 26.6 | 30.4 | 29.9 | **31.8** |
| SVAMP | 5.0 | 5.3 | 5.3 | 6.3 | **7.3** |
| COPA | 70.0 | 64.0 | 66.0 | 67.0 | **71.0** |
| PIQA | 71.9 | 68.1 | 70.8 | 70.7 | **72.6** |
| HellaSwag | 54.8 | 41.9 | 49.1 | 49.2 | **50.0** |
| WinoGrande | 57.9 | 52.6 | 54.3 | 54.0 | **54.9** |
| LAMBADA | 52.2 | 38.5 | **46.1** | **46.1** | 46.0 |
| BoolQ | 50.6 | 55.7 | 60.4 | 60.2 | **61.6** |
| LSAT-RC | 27.2 | 24.6 | 25.7 | 24.3 | **28.7** |
| LSAT-LR | 23.7 | **28.2** | 26.9 | 25.1 | 26.9 |
| SAT-EN | 26.7 | 22.8 | 25.7 | **27.7** | 25.7 |
| Average | 43.1 | 39.1 | 41.9 | 42.0 | **43.2** |

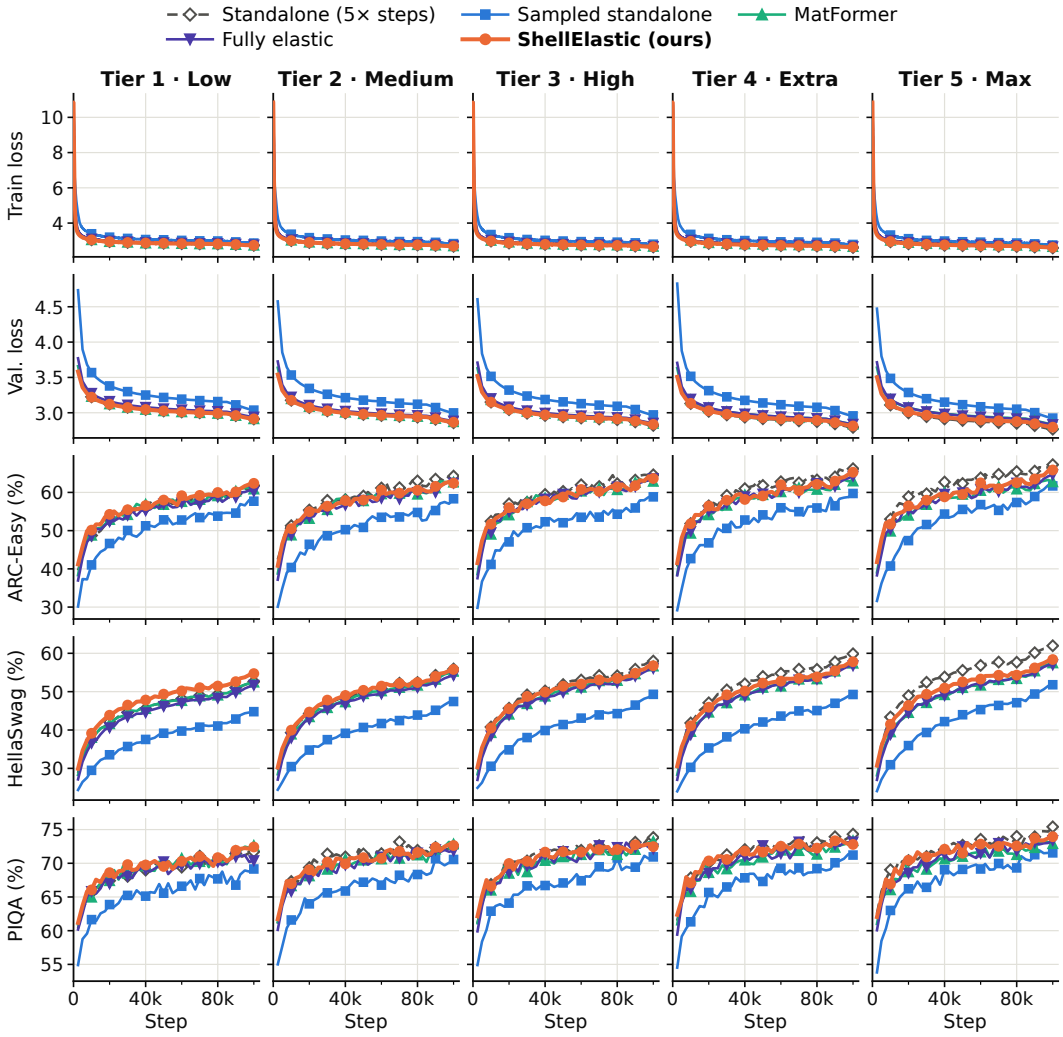


Figure 9: **Training curves at the 675M–1.69B scale** (Table 2). Columns are the five tiers, from Tier 1 (Low, about 675M parameters) to Tier 5 (Max, about 1.69B); rows are the train loss, the validation loss and 5-shot accuracy on ARC-Easy, HellaSwag and PIQA. Each line is a single run; conventions in Appendix H.

Table 22: **Per-task results at the 675M–1.69B scale, Tier 1 (Low, about 675M parameters).** Sub-network width 832 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 4096 at $D_{\text{model}} = 1024$ for MatFormer, and 1024 for SHELLELASTIC. Single runs.

| **Tier 1 · Low** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 676.6 | 676.6 | 673.8 | 676.6 | 673.8 |
| Train loss ↓ | 2.7335 | 2.8643 | 2.7365 | 2.7638 | **2.7347** |
| Val. loss ↓ | 2.9118 | 3.0370 | 2.9140 | 2.9403 | **2.9135** |
| *5-shot accuracy (%) ↑* | | | | | |
| MMLU | 25.2 | 23.6 | **24.1** | 23.3 | **24.1** |
| ARC-Easy | 61.8 | 57.7 | 60.9 | 60.3 | **62.4** |
| ARC-Challenge | 31.4 | 29.6 | **32.3** | 30.2 | 32.2 |
| SVAMP | 6.3 | 6.3 | **6.7** | 4.7 | 4.7 |
| COPA | 70.0 | 64.0 | 67.0 | **71.0** | 70.0 |
| PIQA | 71.7 | 69.2 | **72.6** | 70.4 | 72.4 |
| HellaSwag | 52.6 | 44.8 | 52.5 | 51.7 | **54.7** |
| WinoGrande | 57.1 | 50.8 | 56.7 | 56.9 | **58.2** |
| LAMBADA | 47.1 | 43.0 | 49.7 | 44.0 | **51.6** |
| BoolQ | 57.3 | 52.4 | 60.6 | 55.7 | **61.9** |
| LSAT-RC | 29.1 | **29.5** | 23.9 | 26.9 | 26.1 |
| LSAT-LR | 24.9 | 28.4 | 27.5 | **29.0** | 24.9 |
| SAT-EN | 29.1 | 23.8 | 22.3 | 22.8 | **25.2** |
| Average | 43.4 | 40.2 | 42.8 | 42.1 | **43.7** |

Table 23: **Per-task results at the 675M–1.69B scale, Tier 2 (Medium, about 882M parameters).** Sub-network width 960 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 6144 at $D_{\text{model}} = 1024$ for MatFormer, and 1280 for SHELLELASTIC. Single runs.

| **Tier 2 · Medium** Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|
| Params (M) | 885.8 | 885.8 | 875.2 | 885.8 | 878.0 |
| Train loss ↓ | 2.6791 | 2.8253 | 2.6938 | 2.7067 | **2.6865** |
| Val. loss ↓ | 2.8587 | 2.9980 | 2.8720 | 2.8840 | **2.8657** |
| *5-shot accuracy (%) ↑* | | | | | |
| MMLU | 23.8 | 24.0 | **24.6** | 24.1 | 23.6 |
| ARC-Easy | 64.3 | 58.3 | **62.7** | 62.5 | 62.4 |
| ARC-Challenge | 33.8 | 28.2 | **33.4** | 31.6 | 32.9 |
| SVAMP | 6.0 | **7.0** | **7.0** | 5.0 | 6.0 |
| COPA | 73.0 | 63.0 | 69.0 | **73.0** | 65.0 |
| PIQA | 72.6 | 70.6 | **72.9** | 71.9 | 72.6 |
| HellaSwag | 55.9 | 47.4 | 55.3 | 54.2 | **55.8** |
| WinoGrande | 59.0 | 52.7 | 57.5 | 58.2 | **58.8** |
| LAMBADA | 52.0 | 43.8 | 51.2 | 48.2 | **52.4** |
| BoolQ | 55.0 | 59.0 | 62.6 | 55.8 | **62.8** |
| LSAT-RC | 26.1 | 23.1 | 20.9 | **27.6** | 23.5 |
| LSAT-LR | 25.3 | **27.8** | 25.5 | 27.5 | 27.3 |
| SAT-EN | 23.8 | **24.8** | 20.9 | **24.8** | **24.8** |
| Average | 43.9 | 40.8 | 43.3 | 43.4 | **43.7** |

Table 24: **Per-task results at the 675M–1.69B scale, Tier 3 (High, about 1.12B parameters).** Sub-network width 1088 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 8704 at $D_{\text{model}} = 1024$ for MatFormer, and 1536 for SHELLELASTIC. Single runs.

| **Tier 3 · High** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 1123.2 | 1123.2 | 1126.8 | 1123.2 | 1117.7 |
| Train loss ↓ | 2.6424 | 2.7994 | 2.6730 | 2.6788 | **2.6550** |
| Val. loss ↓ | 2.8212 | 2.9696 | 2.8487 | 2.8539 | **2.8351** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 25.3 | 22.9 | **24.3** | **24.3** | 24.2 |
| ARC-Easy | 64.6 | 58.8 | 62.9 | **63.6** | **63.6** |
| ARC-Challenge | 34.3 | 30.1 | 33.4 | 32.8 | **33.8** |
| SVAMP | 6.3 | 5.3 | **7.0** | 5.0 | **7.0** |
| COPA | 72.0 | 68.0 | 69.0 | **71.0** | 67.0 |
| PIQA | 73.8 | 70.9 | **73.2** | 72.1 | 72.5 |
| HellaSwag | 58.0 | 49.3 | 56.7 | 56.1 | **56.8** |
| WinoGrande | 59.7 | 54.9 | **59.4** | 58.7 | 58.6 |
| LAMBADA | 52.7 | 45.8 | **52.7** | 50.2 | 52.4 |
| BoolQ | 61.8 | 54.2 | 61.3 | 57.9 | **62.1** |
| LSAT-RC | 22.4 | **26.1** | 23.5 | 25.4 | 22.8 |
| LSAT-LR | 24.5 | 28.0 | 23.3 | **28.8** | 24.7 |
| SAT-EN | 25.7 | **25.7** | 18.0 | 25.2 | **25.7** |
| Average | 44.7 | 41.6 | 43.4 | **43.9** | **43.9** |

Table 25: **Per-task results at the 675M–1.69B scale, Tier 4 (Extra, about 1.39B parameters).** Sub-network width 1216 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 11520 at $D_{\text{model}} = 1024$ for MatFormer, and 1792 for SHELLELASTIC. Single runs.

| **Tier 4 · Extra** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 1388.5 | 1388.5 | 1403.7 | 1388.5 | 1393.1 |
| Train loss ↓ | 2.6122 | 2.7838 | 2.6617 | 2.6621 | **2.6323** |
| Val. loss ↓ | 2.7933 | 2.9557 | 2.8385 | 2.8384 | **2.8131** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 23.7 | 24.6 | 24.2 | **24.8** | 24.2 |
| ARC-Easy | 66.2 | 59.7 | 63.0 | 64.3 | **65.3** |
| ARC-Challenge | 35.1 | 30.0 | 32.8 | **33.6** | 32.9 |
| SVAMP | 6.7 | 2.7 | **8.0** | 3.7 | 6.0 |
| COPA | 73.0 | 64.0 | 72.0 | **73.0** | 68.0 |
| PIQA | 74.3 | 71.2 | **73.3** | 73.2 | 72.8 |
| HellaSwag | 59.9 | 49.3 | 57.4 | 56.8 | **57.8** |
| WinoGrande | 59.7 | 54.2 | **59.3** | 58.8 | 59.2 |
| LAMBADA | 55.2 | 46.6 | **53.3** | 51.6 | 52.5 |
| BoolQ | 64.6 | 56.3 | 61.3 | 59.2 | **62.4** |
| LSAT-RC | 24.6 | 25.0 | 23.1 | **26.1** | 24.3 |
| LSAT-LR | 28.4 | 22.7 | 25.9 | **29.2** | 25.1 |
| SAT-EN | 29.1 | **27.2** | 18.0 | 20.9 | 25.7 |
| Average | 46.2 | 41.0 | 44.0 | 44.2 | **44.3** |

Table 26: **Per-task results at the 675M–1.69B scale, Tier 5 (Max, about 1.69B parameters).** Sub-network width 1344 for Standalone, sampled standalone and the fully elastic model, FFN hidden width 14592 at $D_{\text{model}} = 1024$ for MatFormer, and 2048 for SHELLELASTIC. Single runs.

| **Tier 5 · Max** | Standalone† | Sampled standalone | MatFormer | Fully elastic | **SHELLELASTIC** |
|---|---|---|---|---|---|
| Params (M) | 1682.0 | 1682.0 | 1705.6 | 1682.0 | 1704.2 |
| Train loss ↓ | 2.5859 | 2.7509 | 2.6585 | 2.6549 | **2.6152** |
| Val. loss ↓ | 2.7674 | 2.9242 | 2.8378 | 2.8334 | **2.7959** |
| *5-shot accuracy (%)* ↑ | | | | | |
| MMLU | 25.2 | **24.7** | 24.2 | 24.3 | 24.6 |
| ARC-Easy | 67.3 | 61.7 | 62.8 | 64.6 | **65.9** |
| ARC-Challenge | 36.6 | 32.4 | 32.6 | **34.1** | 33.9 |
| SVAMP | 5.0 | 4.7 | **6.3** | 5.3 | 6.0 |
| COPA | 73.0 | 66.0 | **72.0** | **72.0** | 66.0 |
| PIQA | 75.4 | 71.5 | 72.9 | 73.4 | **73.9** |
| HellaSwag | 62.0 | 51.8 | 57.5 | 57.0 | **58.4** |
| WinoGrande | 60.5 | 55.0 | 59.2 | **60.2** | 59.2 |
| LAMBADA | 56.9 | 47.6 | **53.5** | 52.2 | 52.4 |
| BoolQ | 63.0 | 61.4 | 60.6 | 59.3 | **62.5** |
| LSAT-RC | 25.0 | **27.2** | 20.1 | **27.2** | 22.4 |
| LSAT-LR | 22.7 | **28.6** | 24.1 | 28.0 | 27.8 |
| SAT-EN | 26.7 | **29.1** | 20.4 | 21.4 | 25.2 |
| Average | 46.1 | 43.2 | 43.6 | **44.6** | 44.5 |

# I ADDITIONAL TRANSFER RESULTS

Figures 10–12 repeat the sweeps of Figure 2 for each of the five sub-networks of MatFormer, the fully elastic model and SHELLELASTIC. Lines, stars and Δ val. loss are as in Figure 2, with the loss of each sub-network in place of the mean over the five; the proxy line is repeated in every column, and each row has the same y-range in all three figures. Two observations emerge:

- **$\mu$-ELASTIC transfers to every sub-network.** Under $\mu$-ELASTIC, the optimal $\eta_0$ of every sub-network lies between $10^{-3}$ and $3.3 \times 10^{-3}$ and the optimal $\sigma_0$ at $3.3 \times 10^{-3}$ or $10^{-2}$, and the base values of Section 4.1 are within 0.005 nats of the optimum of every sub-network at the 119M–285M and 375M–906M scales.
- **The gap between SP and $\mu$-ELASTIC is largest at the narrow widths.** The gap is larger at the smaller sub-networks, for example 0.022 nats at Low and 0.007 at Max for SHELLELASTIC in the $\eta_0$ sweep, as in Table 3.

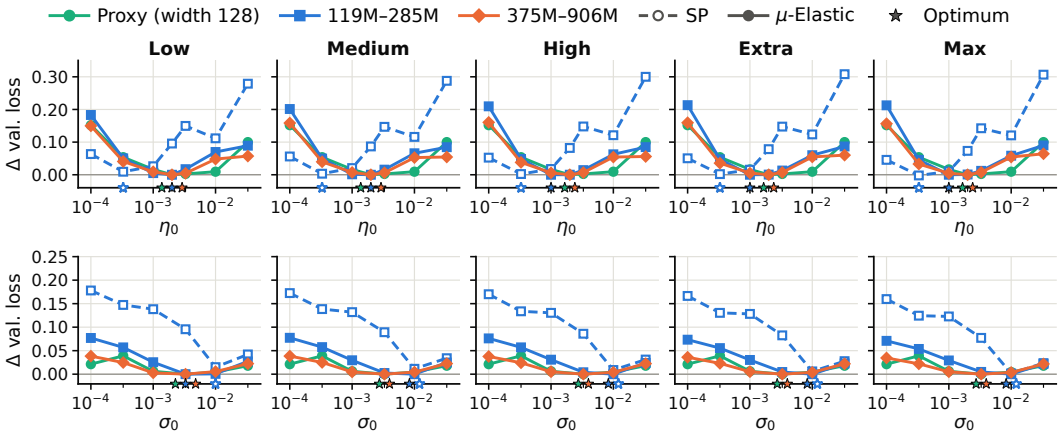


Figure 10: **Per-sub-network sweeps for MatFormer.** The sweeps of Figure 2 for each sub-network, from the smallest (Low) to the largest (Max; widths in Table 6), with the loss of each sub-network in place of the mean over the five.

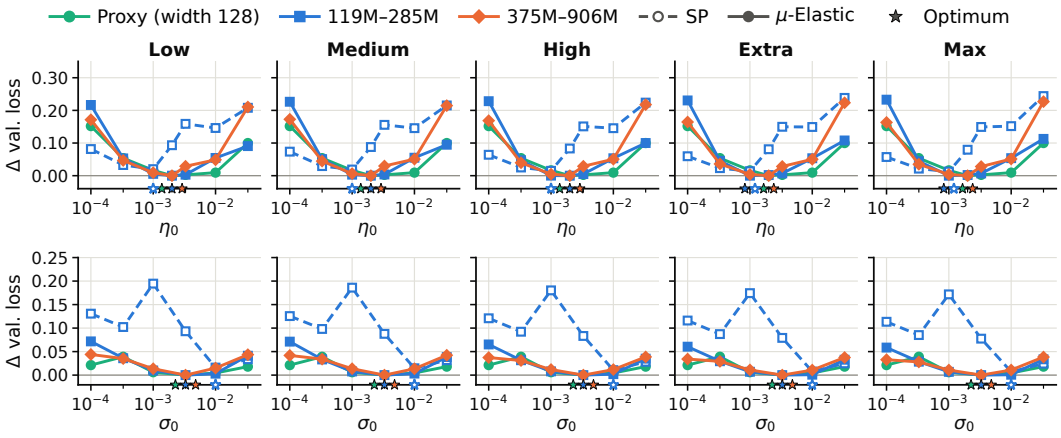


Figure 11: **Per-sub-network sweeps for the fully elastic model**, as in Figure 10.

**Weight decay.** Figure 13 sweeps the weight decay $\lambda_0$ over seven values for the proxy and for SHELLELASTIC at the 119M–285M scale, with $\eta_0$ and $\sigma_0$ at the base values of Section 4.1, and Figure 14 shows each sub-network. Every point is one run, separate from the runs of Figure 2, except the base value $\lambda_0 = 10^{-2}$: it is the base configuration of Figure 2 and uses the same runs (mean of three; SP: two). Two observations emerge:

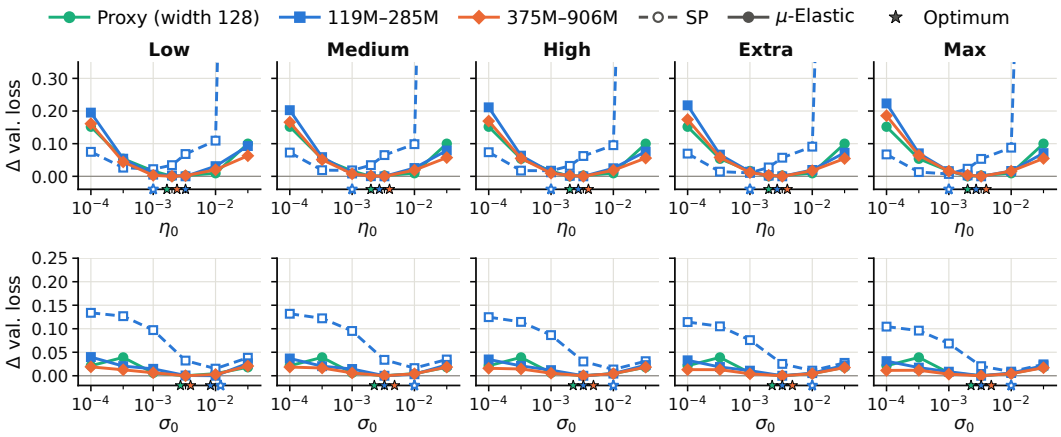


Figure 12: **Per-sub-network sweeps for SHELLELASTIC**, as in Figure 10.

- **Weight decay transfers under $\mu$-ELASTIC.** From $\lambda_0 = 10^{-3}$ to $10^{-1}$, the mean loss of SHELLELASTIC under $\mu$-ELASTIC varies by at most 0.003 nats, the same flat range as the proxy's, and both rise from $3.3 \times 10^{-1}$ on; the base value is within 0.001 nats of the best $\mu$-ELASTIC value.
- **$\mu$-ELASTIC is lower than SP at every swept value.** From $3.3 \times 10^{-3}$ to $10^{-1}$, where SP is also flat (within 0.009 nats), the gap is 0.018–0.028 nats in the mean and again largest at the narrow widths, 0.023–0.032 nats at Low against 0.010–0.020 at Max. SP also degrades at $10^{-3}$, where the gap grows to 0.039 nats, and the gap shrinks to 0.010 and 0.005 nats at $3.3 \times 10^{-1}$ and 1, where both degrade.

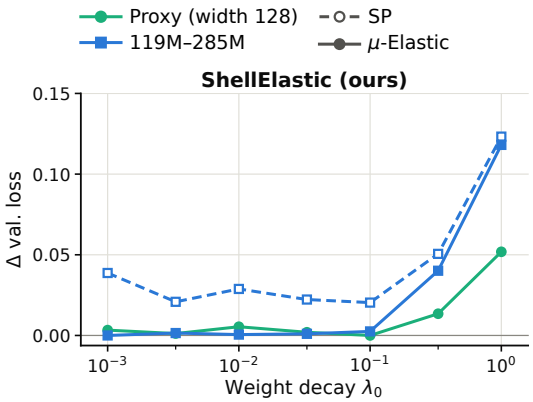


Figure 13: **Weight-decay sweeps.** Sweeps of $\lambda_0$ for the proxy under $\mu$P and for SHELLELASTIC at the 119M–285M scale under $\mu$-ELASTIC (solid) and SP (dashed, open markers); $\Delta$ val. loss is as in Figure 2.

## J ARCHITECTURE COMPARISON AND SEARCH

This appendix complements Section 4.3: Appendix J.1 gives the per-task results and training curves of StairFormer, and Appendix J.2 details the architecture search at equal parameters and width count.

### J.1 COMPARISON WITH STAIRFORMER

StairFormer (Trost et al., 2026) is trained with the recipe of SHELLELASTIC (Section 4.1): the same data, steps, learning rate, initialization and $\mu$-ELASTIC, and, like SHELLELASTIC, all sub-networks

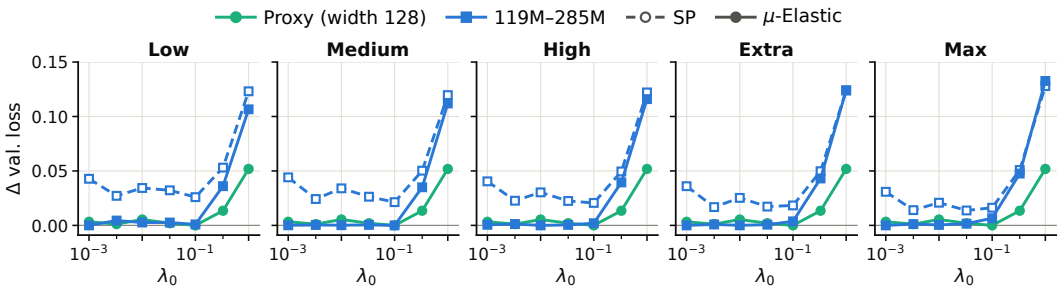


Figure 14: **Per-sub-network weight-decay sweeps.** The sweeps of Figure 13 for each sub-network of SHELLELASTIC, from the smallest (Low) to the largest (Max); lines and Δ val. loss as there, with the loss of each sub-network in place of the mean over the five, and one y-range for all panels.

Table 27: **Per-task results of StairFormer and SHELLELASTIC at every tier** of the 119M–285M scale, with the rows of Appendix H. Stair.: StairFormer (one run). Ours: SHELLELASTIC (mean over three runs; standard deviations in Tables 12–16). Bold marks the better entry of each pair (ties both bold). Only Tier 5 is parameter-matched.

| | Tier 1 · Low | | Tier 2 · Medium | | Tier 3 · High | | Tier 4 · Extra | | Tier 5 · Max | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Stair. | **Ours** | Stair. | **Ours** | Stair. | **Ours** | Stair. | **Ours** | Stair. | **Ours** |
| Params (M) | 32.6 | 118.8 | 76.9 | 152.7 | 132.8 | 190.8 | 200.4 | 233.1 | 279.6 | 279.6 |
| Train loss ↓ | 3.6874 | **3.2305** | 3.4224 | **3.1641** | 3.2764 | **3.1229** | 3.1871 | **3.0941** | 3.1258 | **3.0713** |
| Val. loss ↓ | 3.8836 | **3.4086** | 3.6004 | **3.3375** | 3.4519 | **3.2944** | 3.3588 | **3.2641** | 3.2949 | **3.2403** |
| *5-shot accuracy (%)* ↑ | | | | | | | | | | |
| MMLU | **25.1** | 24.7 | **25.1** | 23.8 | **25.1** | 24.0 | **25.5** | 24.4 | **25.3** | 24.2 |
| ARC-Easy | 35.7 | **45.0** | 41.3 | **45.8** | 41.8 | **47.1** | 44.0 | **47.6** | 45.7 | **48.0** |
| ARC-Challenge | 21.5 | **23.3** | 22.4 | **23.5** | 22.3 | **23.6** | 22.5 | **24.0** | 23.2 | **24.3** |
| SVAMP | **6.0** | 5.7 | 5.3 | **6.3** | **7.3** | 6.3 | **7.0** | 6.7 | **8.3** | 6.4 |
| COPA | 55.0 | **59.7** | 57.0 | **61.7** | 60.0 | **61.7** | 56.0 | **63.0** | 59.0 | **63.3** |
| PIQA | 57.8 | **62.1** | 60.2 | **63.7** | 61.8 | **64.6** | 63.1 | **64.8** | 64.0 | **65.2** |
| HellaSwag | 27.0 | **32.4** | 29.7 | **33.2** | 31.1 | **33.9** | 32.2 | **34.6** | 33.6 | **35.1** |
| WinoGrande | **52.6** | 50.7 | **51.9** | 49.3 | **51.8** | 50.0 | **52.2** | 49.8 | **50.4** | 49.8 |
| LAMBADA | 15.0 | **25.9** | 23.3 | **29.1** | 25.3 | **29.9** | 26.8 | **30.2** | 27.4 | **30.4** |
| BoolQ | 47.0 | **55.1** | 48.4 | **57.2** | 53.2 | **55.3** | 51.1 | **54.5** | 51.7 | **52.3** |
| LSAT-RC | **25.4** | 24.8 | 23.1 | **25.6** | 22.4 | **26.0** | 25.4 | **26.4** | **28.0** | 25.6 |
| LSAT-LR | 25.7 | **26.2** | 26.5 | **29.2** | 24.1 | **26.8** | 24.5 | **27.6** | 24.1 | **27.6** |
| SAT-EN | **25.7** | 25.6 | **27.2** | **27.2** | 24.3 | **26.2** | 24.3 | **26.5** | 23.8 | **26.9** |
| Average | 32.3 | **35.5** | 34.0 | **36.6** | 34.7 | **36.6** | 35.0 | **36.9** | 35.7 | **36.9** |

at every step. It differs only in architecture: five equal 208-wide groups with $d_{\text{head}} = 104$, so that each group holds two complete heads, a cumulative prefix norm (Equation 25), and one readout shared by all sub-networks. Table 27 gives its per-task results at every tier, and Figure 15 its training curves next to the methods of Table 2 other than sampled standalone. Two observations emerge:

- **Only Max is parameter-matched.** StairFormer's Max and that of SHELLELASTIC both have 279.6M parameters, but at Tiers 1–4, StairFormer's sub-networks are 14–73% smaller (32.6–200.4M against 118.8–233.1M), so these pairs compare sub-networks of different sizes; Figure 3 compares the validation loss and average 5-shot accuracy against parameter count.
- **At Max, SHELLELASTIC is better on most tasks.** At the matched size, SHELLELASTIC is better on 9 of the 13 tasks and worse on 4 (MMLU, SVAMP, WinoGrande and LSAT-RC).

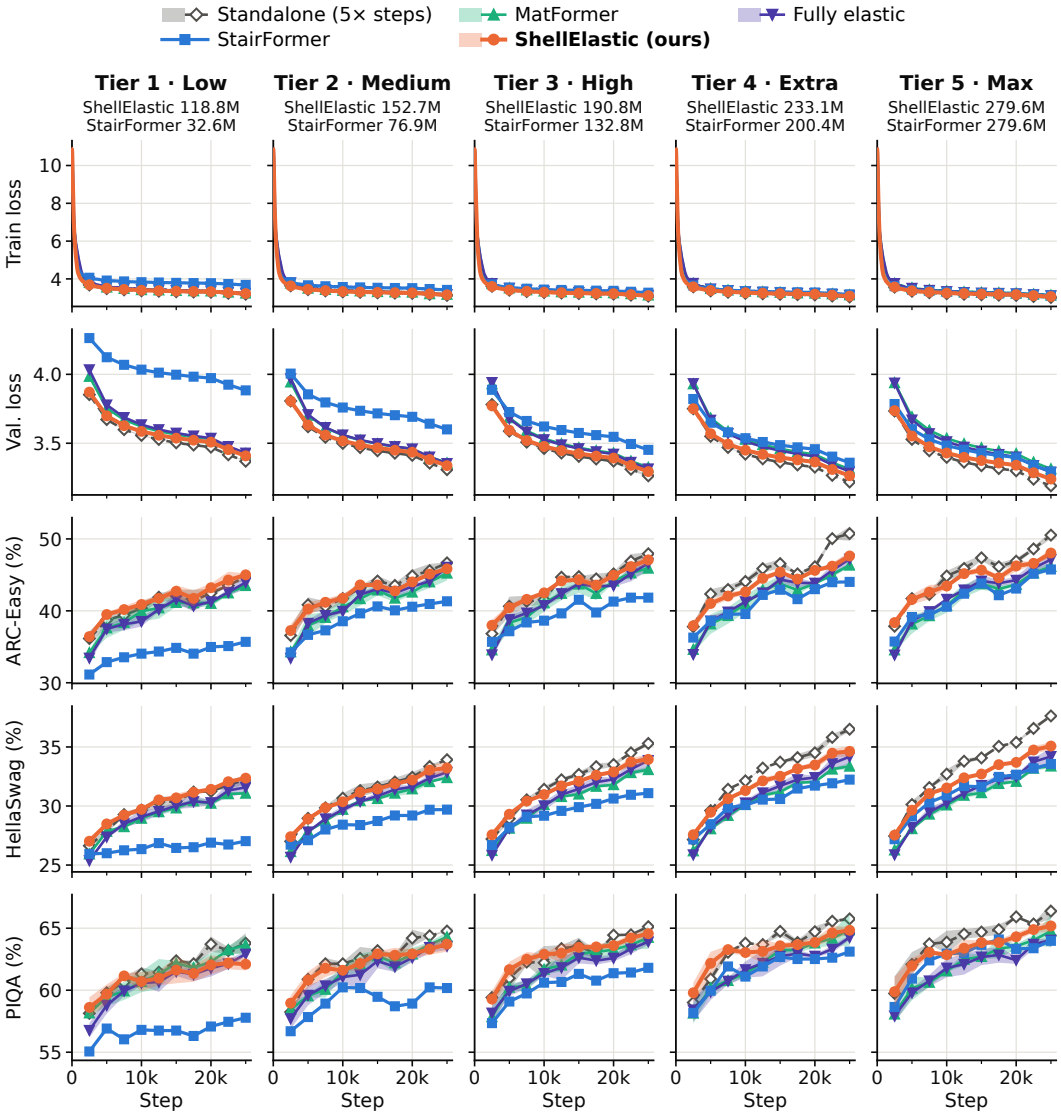


Figure 15: **Training curves of StairFormer and the methods of Table 2** at the 119M–285M scale, except sampled standalone, in the format of Figure 7; column titles give the parameters of SHELLELASTIC and StairFormer. StairFormer is a single run; the other lines are means over three runs with $\pm 1$ standard deviation bands.

### J.2 Architecture search at equal parameters and width count

The runs of Figures 3c and 17 train every supported width at every step with the default recipe and change only $\mathcal{S}$. A supported set fixes how many of the $M^2$ blocks of size $g \times g$ a hidden matrix keeps: group $\mathcal{G}_k$ has $s_k - s_{k-1}$ block rows, each reading $s_k$ input blocks, so

$$B(\mathcal{S}) = \sum_{k=1}^{K} (s_k - s_{k-1})\, s_k. \tag{24}$$

We compare two families of supported sets (Figure 16): five supported widths at $B = 42$, the block count of the default $\mathcal{S} = \{4, \dots, 8\}$, and six supported widths at $B = 39$. Every set contains exactly one group wider than $g$, and the sets of a family differ only in where this group sits, from the last shell to the core. A width at or above the wide group has the same number of blocks wherever the group sits, so its parameter count is the same in every set of the family; a width below it keeps only the lower-triangle blocks and has fewer parameters (63.9M at 384, 93.7M at 512, 127.6M at 640; marked † in Table 28). $\mathcal{S} = \{4, \dots, 8\}$ is the default configuration, so its row is the mean of the three runs of Table 2 at the 119M–285M scale (standard deviation 0.0009 to 0.0027 per width); the other sets are single runs. The average in Table 28 is taken over the supported widths of each set, so it covers different widths in different rows, and the core placements have no width below the core.

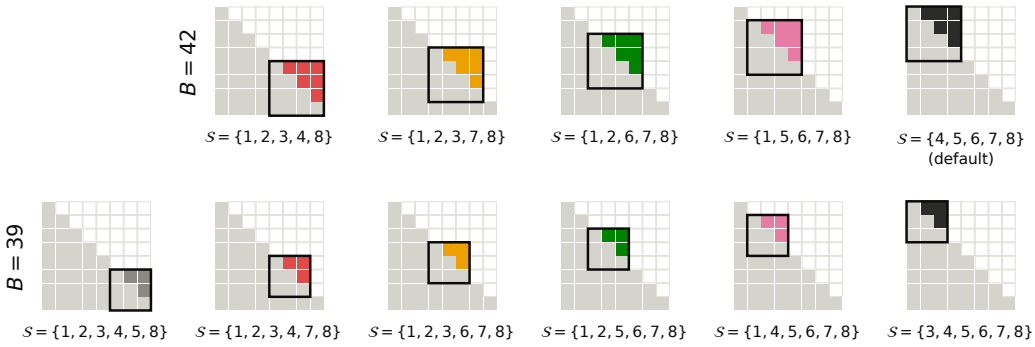


Figure 16: **Block masks of the supported sets in Figures 3c and 17** ($M = 8$, $g = 128$). Cell $(i, j)$ is the $g \times g$ block of every hidden matrix from input block $j$ to output block $i$. Gray: lower-triangle blocks kept by every set; colored: blocks above the diagonal, all inside the one group wider than $g$ (outlined). Each row has the same $B$ and moves the wide group from the last shell (left) to the core (right).

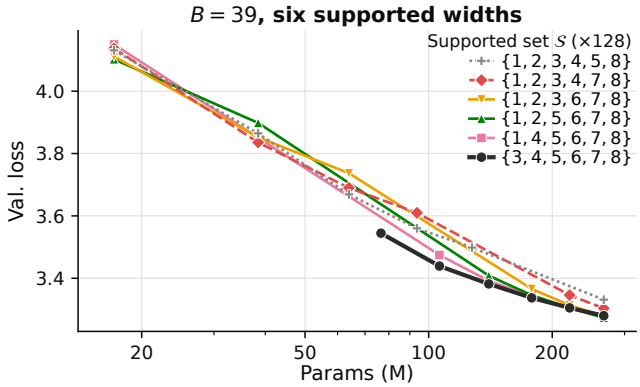


Figure 17: As Figure 3c, at $B = 39$ with six supported widths (core placement $\mathcal{S} = \{3, \dots, 8\}$, black). Every set is a single run.

Table 28: **Validation loss of every supported width in the architecture search**, at step 25,000, as the one group wider than $g$ moves from the last shell (first row of each block) to the core (last row, bold). Params: non-zero parameters (M) of each width; †a width below the wide group, which has fewer parameters. Avg.: mean over the supported widths of each set. Bold marks the lowest value in each column.

| $\mathcal{S}$ | 128 | 256 | 384 | 512 | 640 | 768 | 896 | 1024 | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| *$B = 42$, 5 supported widths* | | | | | | | | | |
| Params | 17.1 | 38.4 | 63.9 | 118.8 | 152.7 | 190.8 | 233.1 | 279.6 | |
| $\{1, 2, 3, 4, 8\}$ | 4.1178 | **3.8183** | **3.6738** | 3.6023† | – | – | – | 3.2698 | 3.6964 |
| $\{1, 2, 3, 7, 8\}$ | 4.1037 | 3.8310 | 3.7224 | – | – | – | 3.2910 | 3.2503 | 3.6397 |
| $\{1, 2, 6, 7, 8\}$ | **4.0895** | 3.8832 | – | – | – | 3.3193 | 3.2721 | **3.2400** | 3.5608 |
| $\{1, 5, 6, 7, 8\}$ | 4.1154 | – | – | – | 3.3657 | 3.3068 | 3.2687 | 3.2409 | 3.4595 |
| **$\{4, 5, 6, 7, 8\}$** | – | – | – | **3.4086** | **3.3375** | **3.2944** | **3.2641** | 3.2403 | **3.3090** |
| *$B = 39$, 6 supported widths* | | | | | | | | | |
| Params | 17.1 | 38.4 | 76.5 | 106.2 | 140.1 | 178.2 | 220.5 | 267.0 | |
| $\{1, 2, 3, 4, 5, 8\}$ | 4.1310 | 3.8646 | 3.6689† | 3.5601† | 3.4983† | – | – | 3.3314 | 3.6757 |
| $\{1, 2, 3, 4, 7, 8\}$ | 4.1371 | **3.8356** | 3.6898† | 3.6098† | – | – | 3.3460 | 3.3035 | 3.6537 |
| $\{1, 2, 3, 6, 7, 8\}$ | 4.1112 | 3.8520 | 3.7369† | – | – | 3.3659 | 3.3134 | 3.2775 | 3.6095 |
| $\{1, 2, 5, 6, 7, 8\}$ | **4.1022** | 3.8981 | – | – | 3.4082 | 3.3456 | **3.3047** | **3.2734** | 3.5554 |
| $\{1, 4, 5, 6, 7, 8\}$ | 4.1499 | – | – | 3.4746 | 3.3919 | 3.3419 | 3.3057 | 3.2778 | 3.4903 |
| **$\{3, 4, 5, 6, 7, 8\}$** | – | – | **3.5445** | **3.4391** | **3.3818** | **3.3371** | 3.3050 | 3.2798 | **3.3812** |

## K Ablation studies

All ablations use the supernet of Section 4.1 ($g = 128$, $\mathcal{S} = \{4, \ldots, 8\}$) at the 119M–285M scale and change one setting of its recipe at a time. Tables 29 and 30 list all results, numbered as the paragraphs below; each block of rows ends with the default.

- **Default and run-to-run variation.** The default is not re-run: its rows are the mean of the three SHELLELASTIC runs of Table 2, whose standard deviation is at most 0.003 nats per tier and 0.002 nats on the mean validation loss, and 0.32 points on the mean 5-shot accuracy; every other row is one run. A further run of the default on H100 GPUs, which only adds the logging of ablation (8) and is not part of this mean, is within 0.003 nats of it at every tier, so we treat smaller differences as run-to-run variation.
- **Hardware and cost.** The default runs used A100 GPUs with activation checkpointing on every other block, as did ablations (11) and (12), whereas ablations (1) to (10) used H100 GPUs without it. Step times and peak memory are compared between runs on the same GPUs with the same activation checkpointing; on H100 GPUs, the reference is the loss-weight runs of (2) (37.1 GB peak memory), which do the same computation as the default, unlike the logging run of (8).

**(1) Selected widths per step.** Besides the default (All), which selects all five widths at every step, we test four rules for the selected set $\mathcal{A}_\tau$ of Equation 8:

- *Sandwich*: widths 512 and 1024 at every step and one of 640 to 896 uniformly at random;
- *Max + random*: width 1024 and one of the other four;
- *Random*: one of the five widths uniformly at random, as MatFormer and the fully elastic model do;
- *Max only*: width 1024 alone.

**Observation:** The loss of each width follows how often it is trained, and All gives the lowest mean loss. Sandwich and Max + random leave the loss of width 1024 within 0.001 nats but raise the loss of the widths they select less often: Sandwich lowers the loss of Low by 0.007 nats and raises those of Medium to Extra by 0.018 to 0.024 nats, and Max + random raises Low to Extra by 0.031 to

Table 29: **Ablations (1)–(6) of SHELLELASTIC at the 119M–285M scale.** Each block changes one setting of the default recipe and ends with the default (mean of three runs); every other row is one run. Val. loss is taken at step 25,000 for each tier (Low to Max: widths 512 to 1024), and 5-shot acc. averages the 13 downstream tasks over the five tiers. Bold marks the best value of each column within a block, and "–" a width whose readout is not trained (and the averages of its row).

| | Val. loss ↓ | | | | | | 5-shot acc. ↑ |
|---|---|---|---|---|---|---|---|
| Setting | Low | Medium | High | Extra | Max | Avg. | Avg. |
| *(1) Selected widths per step $\mathcal{A}_\tau$* | | | | | | | |
| Sandwich: 512, 1024 and one of 640–896 | **3.402** | 3.362 | 3.318 | 3.282 | 3.240 | 3.321 | 36.1 |
| Random: one of the five widths | 3.431 | 3.379 | 3.354 | 3.341 | 3.336 | 3.368 | 34.8 |
| Max only: width 1024 | – | – | – | – | **3.227** | – | – |
| Max + random: 1024 and one of 512–896 | 3.494 | 3.408 | 3.345 | 3.296 | 3.240 | 3.356 | 35.5 |
| All: all five widths (default) | 3.409 | **3.338** | **3.294** | **3.264** | 3.240 | **3.309** | **36.5** |
| *(2) Loss weights $\omega_s$ (widths 512 to 1024)* | | | | | | | |
| $(1, 1, 1, 1, 1)$ | 3.389 | **3.328** | **3.292** | 3.268 | 3.252 | 3.306 | 36.5 |
| $(1, 0.5, 0.3, 0.2, 2)$ | **3.380** | 3.331 | 3.298 | 3.269 | 3.241 | **3.304** | **36.6** |
| $(1, 1, 1, 1, 2)$ | 3.396 | 3.332 | 3.293 | 3.267 | 3.249 | 3.307 | 36.3 |
| $(1, 1, 1, 1, 6)$ | 3.424 | 3.344 | 3.298 | **3.264** | **3.237** | 3.313 | 36.2 |
| $(1, 1, 1, 1, 4)$ (default) | 3.409 | 3.338 | 3.294 | **3.264** | 3.240 | 3.309 | 36.5 |
| *(3) Trunk normalization* | | | | | | | |
| Cumulative prefix | **3.406** | 3.338 | 3.295 | **3.264** | 3.241 | **3.309** | 36.1 |
| Block-local over every $g$ coordinates | 3.409 | **3.337** | 3.295 | 3.265 | 3.241 | 3.310 | **36.7** |
| Group-local over $\mathcal{G}_k$ (default) | 3.409 | 3.338 | **3.294** | **3.264** | **3.240** | **3.309** | 36.5 |
| *(4) Terminal norm $N_s^{\text{out}}$* | | | | | | | |
| Group-local, as the trunk | 3.412 | 3.341 | 3.297 | 3.266 | 3.242 | 3.312 | 35.4 |
| Over all $sg$ coordinates (default) | **3.409** | **3.338** | **3.294** | **3.264** | **3.240** | **3.309** | **36.5** |
| *(5) Readout* | | | | | | | |
| Shared $U$ and terminal gain | 3.415 | 3.345 | 3.300 | 3.270 | 3.245 | 3.315 | 35.6 |
| Separate $U_s$ and terminal gains (default) | **3.409** | **3.338** | **3.294** | **3.264** | **3.240** | **3.309** | **36.5** |
| *(6) In-place distillation from width 1024* | | | | | | | |
| Forward KL, weight 0.1 | 3.405 | 3.335 | 3.294 | 3.264 | 3.241 | 3.308 | 36.4 |
| Forward KL, weight 0.5 | 3.392 | 3.329 | 3.293 | 3.267 | 3.248 | 3.306 | **36.6** |
| Forward KL, weight 1 | **3.375** | **3.318** | **3.287** | 3.265 | 3.248 | **3.299** | 36.1 |
| Reverse KL, weight 0.5 | 3.387 | 3.326 | 3.289 | **3.263** | 3.244 | 3.302 | 35.9 |
| Symmetric KL, weight 0.5 | 3.388 | 3.326 | 3.290 | **3.263** | 3.244 | 3.302 | 36.0 |
| None (default) | 3.409 | 3.338 | 3.294 | 3.264 | **3.240** | 3.309 | 36.5 |

0.086 nats. Random raises every tier, by 0.022 nats at Low and 0.095 nats at Max, and Max only lowers the loss of width 1024 by 0.013 nats but leaves the other widths untrained.

**(2) Loss weights.** With all widths selected at every step, we vary the loss weights $\omega_s$ of Equation 8: the weight of width 1024 rises from 1 to 2, 4 (the default) and 6, and the decaying weights $(1, 0.5, 0.3, 0.2, 2)$, which Equation 8 normalizes, keep the relative weight of width 1024 at $1/2$ and double that of width 512.

**Observation:** The loss weight of a width moves its loss in the same way as how often it is trained. Raising the weight of width 1024 lowers its loss monotonically (3.252, 3.249, 3.240 and 3.237 at weights 1, 2, 4 and 6) and raises the mean loss from 3.306 to 3.313, mostly through Low and Medium. The decaying weights match the default at Max (0.001 nats higher) and lower the mean loss by 0.005 nats, through Low and Medium (0.029 and 0.006 nats lower) at a cost at High and Extra (0.003 and 0.005 nats higher). Training all widths at every step with weights $(1, 1, 1, 1, 4)$ is thus within 0.005 nats of the best trade-off we found.

**(3) Trunk normalization.** We compare three trunk normalizations that all preserve the supported prefixes (Appendix C.2): group-local normalization over each $\mathcal{G}_k$ (the default), block-local normalization over every $g$ coordinates, and a cumulative prefix norm, which normalizes each

Table 30: **Ablations (7)–(12) of SHELLELASTIC at the 119M–285M scale**, in the format of Table 29.

| | Val. loss ↓ | | | | | | 5-shot acc. ↑ |
|---|---|---|---|---|---|---|---|
| Setting | Low | Medium | High | Extra | Max | Avg. | Avg. |
| *(7) Normalization type* | | | | | | | |
| LayerNorm | **3.404** | **3.338** | 3.296 | 3.265 | 3.242 | **3.309** | 35.8 |
| RMSNorm (default) | 3.409 | **3.338** | **3.294** | **3.264** | **3.240** | **3.309** | **36.5** |
| *(8) Ensembles of the five widths* | | | | | | | |
| Learned per-token width router | 3.774 | 3.561 | 3.414 | 3.306 | **3.227** | 3.456 | 35.1 |
| Distillation from the ensemble | **3.390** | **3.327** | **3.293** | 3.267 | 3.247 | **3.305** | **37.0** |
| Distillation from a label-aware mixture | 3.398 | 3.335 | 3.298 | 3.271 | 3.250 | 3.310 | 36.3 |
| None (default) | 3.409 | 3.338 | 3.294 | **3.264** | 3.240 | 3.309 | 36.5 |
| *(9) Stage-residual gradient scaling* | | | | | | | |
| $\gamma = 0$ | 3.419 | 3.374 | 3.347 | 3.326 | 3.410 | 3.375 | 35.3 |
| $\gamma = 0.3$ | **3.381** | **3.332** | 3.299 | 3.276 | 3.249 | **3.307** | 36.6 |
| $\gamma = 0.5$ | 3.394 | 3.336 | 3.297 | 3.270 | 3.245 | 3.308 | 36.0 |
| $\gamma = 0.8$ | 3.403 | 3.336 | 3.295 | 3.266 | 3.242 | 3.308 | **37.0** |
| $\gamma = 1$ (default) | 3.409 | 3.338 | **3.294** | **3.264** | **3.240** | 3.309 | 36.5 |
| *(10) Trunk gradient of widths* 512–896 | | | | | | | |
| Detached | 3.929 | 3.648 | 3.449 | 3.313 | **3.226** | 3.513 | 35.2 |
| Kept (default) | **3.409** | **3.338** | **3.294** | **3.264** | 3.240 | **3.309** | **36.5** |
| *(11) Base width $g$ and head dimension $d_{\text{head}}$* | | | | | | | |
| $g = 64$, $d_{\text{head}} = 64$ | **3.400** | 3.334 | **3.292** | **3.262** | **3.238** | **3.305** | 36.2 |
| $g = 128$, $d_{\text{head}} = 64$ | 3.402 | **3.333** | 3.293 | 3.264 | 3.240 | 3.306 | 36.3 |
| $g = 128$, $d_{\text{head}} = 128$ (default) | 3.409 | 3.338 | 3.294 | 3.264 | 3.240 | 3.309 | **36.5** |
| *(12a) Gradient clipping (global norm)* | | | | | | | |
| 0.5 | 3.409 | 3.339 | 3.296 | 3.266 | 3.242 | 3.310 | 36.3 |
| 1 | 3.415 | 3.343 | 3.299 | 3.268 | 3.244 | 3.314 | 35.9 |
| 2 | **3.404** | **3.334** | **3.293** | **3.263** | 3.241 | **3.307** | **36.6** |
| 4 (default) | 3.409 | 3.338 | 3.294 | 3.264 | **3.240** | 3.309 | 36.5 |
| *(12b) AdamW $\epsilon$* | | | | | | | |
| $10^{-6}$ | 3.410 | 3.341 | 3.301 | 3.274 | 3.255 | 3.316 | 35.7 |
| $10^{-8}$ | **3.406** | **3.335** | **3.292** | **3.263** | **3.240** | **3.307** | 36.0 |
| $10^{-10}$ | **3.406** | 3.337 | 3.296 | 3.266 | 3.243 | 3.309 | 36.4 |
| $10^{-14}$ (default) | 3.409 | 3.338 | 3.294 | 3.264 | **3.240** | 3.309 | **36.5** |
| *(12c) z-loss coefficient $\lambda_z$* | | | | | | | |
| $10^{-3}$ | 3.410 | 3.339 | 3.297 | 3.266 | 3.241 | 3.311 | 35.9 |
| $10^{-2}$ | **3.407** | **3.336** | **3.294** | **3.264** | **3.240** | **3.308** | **36.5** |
| $10^{-4}$ (default) | 3.409 | 3.338 | **3.294** | **3.264** | **3.240** | 3.309 | **36.5** |

block of $g$ coordinates with the statistics of all coordinates up to the end of that block. With $\mathcal{B}_b = \{(b-1)g+1, \ldots, bg\}$ the $b$-th block of $g$ coordinates, all three apply Equation 3 with the statistics of a coordinate $j \in \mathcal{G}_k \cap \mathcal{B}_b$ taken over

$$\mathcal{G}_k \text{ (group-local)}, \qquad \mathcal{B}_b \text{ (block-local)}, \qquad \{1, \ldots, bg\} \text{ (cumulative prefix)}. \tag{25}$$

**Observation:** The three are within $0.001$ nats of each other in mean loss and within $0.003$ nats at every tier. They differ in cost: the prefix norm needs cumulative statistics for every prefix of blocks, and our implementation has fused kernels only for group-local and block-local normalization, so the prefix norm takes $6\%$ more time per step and $23\%$ more memory than group-local normalization, while block-local normalization costs the same.

**(4) Terminal norm.** We compare normalizing all $sg$ coordinates of sub-network $s$ before its readout (the default, Equation 6) with normalizing each group separately, as in the trunk.

**Observation:** The default is better: the loss is $0.003$ to $0.004$ nats lower at Low to High and $0.001$ to $0.002$ nats lower, within run-to-run variation, at Extra and Max ($0.003$ nats on the mean), and the mean 5-shot accuracy is about $1$ point higher ($36.5$ against $35.4$).

**(5) Readout.** We compare separate readouts $U_s$ and terminal-norm gains for every width (the default) with one readout and one gain shared by all widths, of which width $s$ uses the first $sg$ columns (under $\mu$-ELASTIC, with the frozen-plus-delta split of Equation 23).

**Observation:** Separate readouts and gains are better at every tier, by $0.005$ to $0.007$ nats ($0.006$ nats on the mean), and their mean 5-shot accuracy is $0.9$ points higher ($36.5$ against $35.6$).

**(6) In-place distillation.** We distill widths $512$ to $896$ from width $1024$ by adding to the loss of each a weighted KL divergence between its output distribution and that of width $1024$ over the full vocabulary: forward KL with weights $0.1$, $0.5$ and $1$, and reverse and symmetric KL with weight $0.5$.

**Observation:** Distillation lowers the loss of Low to High, leaves that of Extra within $0.004$ nats and raises that of width $1024$. Forward KL with weights $0.1$, $0.5$ and $1$ lowers the loss of Low by $0.004$, $0.017$ and $0.034$ nats and raises that of Max by $0.001$, $0.008$ and $0.008$ nats, and reverse and symmetric KL lower Low by $0.021$ to $0.022$ nats and raise Max by $0.003$ nats. The mean loss falls by up to $0.010$ nats, but the mean 5-shot accuracy stays within $0.6$ points of the default ($35.9$ to $36.6$ against $36.5$). In our implementation, the teacher distributions and the KL terms make a step $2.2$ to $2.6$ times as long and raise peak memory by $12\%$ to $14\%$. As distillation helps the narrower widths only at the expense of the widest one and adds computation and memory, we do not use it.

**(7) Normalization type.** We compare RMSNorm (the default) with LayerNorm with a gain and no bias.

**Observation:** Both give the same mean loss ($3.309$) and differ by at most $0.005$ nats at any tier. We keep RMSNorm, for which our implementation has fused kernels; the LayerNorm run takes $1.6$ times the time per step and about twice the peak memory.

**(8) Ensembles.** One trunk pass yields the output distributions $p_s = \mathrm{softmax}(z_s)$ of all five widths, so combining them costs little beyond the readouts, and nested widths may still err on different tokens. We test three ways to combine them, each a per-token mixture $\sum_s \pi_s p_s$ with non-negative weights $\pi_s$ that sum to one:

- *Fixed mixtures*: a run with the default objective logs, on every training batch, the loss of the uniform mixture, of the best fixed mixture (the weights that minimize the loss of that batch), of a confidence-weighted mixture ($\pi_s$ proportional to the largest probability in $p_s$), and of the most confident readout of each token; as an upper bound, it also logs an oracle that takes, for each token, the readout assigning the highest probability to the target.
- *Learned router*: a two-layer MLP with hidden width $256$ reads the width-$1024$ hidden state after its terminal norm and outputs per-token weights $\pi_s$, starting from uniform weights; the loss becomes the router-weighted sum of the five losses at every token, plus $0.01$ times their mean, which keeps every width trained.
- *Distillation from mixtures*: every width, including width $1024$, adds a forward KL divergence with weight $0.5$ toward a detached teacher, either the mixture with the loss weights $\omega_s$ of Equation 8 or a per-token mixture with $\pi_s \propto p_s(y)^\beta$, which favors the readouts that assign a high probability to the target $y$; $\beta$ is set at every step so that the mean normalized entropy of the weights is $0.7$.

**Observation:** None of these combinations beats width $1024$ alone. Over the last $500$ steps, width $1024$ alone has a loss of $3.062$ nats; the best fixed mixture of the five, fitted on each batch, matches it (within $0.0001$ nats), averaging them uniformly or weighting them by their confidence is $0.03$ nats higher, and picking the most confident readout for each token is $0.065$ nats higher. Only the oracle is lower ($2.889$ nats), but it needs the target. The router collapses onto width $1024$ (weight $1.00$): its Max then improves by $0.013$ nats, as when width $1024$ is trained alone in (1), while the other widths, trained only through the $0.01$ term, lose $0.041$ to $0.365$ nats. Distillation toward either mixture does

not improve width 1024 either: its loss rises by 0.007 and 0.010 nats, while that of Low falls by 0.018 and 0.010 nats, as with in-place distillation in (6), and a step takes 2.9 and 3.3 times as long.

**Outlook:** Combining widths remains promising for elastic models: the oracle is 0.17 nats below width 1024 alone, so the widths are complementary, and a label-free selection better than confidence could turn this into a gain at almost no extra cost.

**(9) Stage-residual gradient scaling.** The logits of width $s_k$ are a sum of one term per group,

$$z_{s_k} = \alpha_{s_k} \sum_{j \le k} h^{(k)}_{\mathcal{G}_j} \big(U_{s_k}\big)^{\top}_{:,\mathcal{G}_j}, \qquad h^{(k)} = N^{\mathrm{out}}_{s_k}\big(x^{(L)}_{:s_k g}\big), \tag{26}$$

so the supernet can be viewed as a sequence of stages: the core gives a base prediction, and each shell adds a correction to the terms of the groups before it, as in residual learning. To train the shells toward this role, we scale by $\gamma$ the gradient that the loss of each width sends into the terms of the narrower groups, through the coordinates of the next narrower sub-network:

$$\frac{\partial \ell_{s_k}}{\partial h^{(k)}_{:s_{k-1}g}} \;\leftarrow\; \gamma \, \frac{\partial \ell_{s_k}}{\partial h^{(k)}_{:s_{k-1}g}}, \qquad k = 2, \dots, K. \tag{27}$$

Widths 512 to 896 have separate readouts, which their losses train in full; for width 1024, the gradient into its readout columns over those coordinates is scaled as well. The term of the width's own shell keeps the full gradient, the forward pass is unchanged, and $\gamma = 1$ is the joint objective of Equation 8. With $\gamma = 0$, the loss of each width reaches the output terms of its own shell only, which then fit what the narrower groups leave unexplained; the representations are still shared, since the block rows of every shell read the coordinates of the narrower groups in every layer, through which the wider losses still train them.

**Observation:** A $\gamma$ between 0.3 and 0.8 lowers the loss of Low by 0.006 to 0.028 nats but raises those of Extra and Max by 0.002 to 0.012 nats and leaves the mean loss within 0.002 nats. With $\gamma = 0$, every tier is worse, Max by 0.170 nats, so that width 1024 has a higher loss than width 896 (3.410 against 3.326): its readout columns and terminal-norm gains over the first 896 coordinates then receive no gradient and stay at their initialization, up to weight decay. We therefore keep $\gamma = 1$.

**Outlook:** Although $\gamma < 1$ does not lower the mean loss here, the stage view matches a property that, among the three elastic families, only SHELLELASTIC has: it can grow after training. No group reads a later one (Equation 2), and every group is normalized on its own (Equation 3), so appending a shell $\mathcal{G}_{K+1}$ to a trained supernet only adds parameters: one block row in every structured projection, which reads all existing coordinates, the group's normalization gains and embedding columns, and a terminal norm and readout for the new width, with the initialization and learning rates that Table 1 assigns to its boundary. The trained weights can stay frozen, so every trained sub-network keeps its exact outputs and remains a prefix of the extended trunk pass, while only the new shell trains, for example on downstream data, as a new stage on top of the frozen supernet. MatFormer and fully elastic models cannot grow this way. MatFormer shares one model width across all sub-networks, so new FFN units write into the hidden states that its trained layers read; a fully elastic model mixes all channels densely and normalizes over the active width, so new channels change the inputs of every trained weight in the wider sub-network. Their trained weights would then operate on inputs they were not trained on, and the wider sub-network could not reuse the hidden states of the trained ones.

**(10) Trunk gradient of the narrower widths.** We detach the trunk from the losses of widths 512 to 896, which then train only their terminal norms, readouts and $\alpha_s$ on a trunk trained by width 1024 alone.

**Observation:** Detaching lowers the loss of Max by 0.014 nats, as training width 1024 alone does in (1), but raises those of Low to Extra by 0.049 to 0.520 nats, so keeping their trunk gradients is much better overall.

**(11) Base width and head dimension.** At the same supported widths 512 to 1024 and the same parameter counts, halving the base width to $g = 64$ ($M = 16$, $\mathcal{S} = \{8, 10, \dots, 16\}$) keeps the 512-wide core and every shell 128 wide, so the groups, their normalization and the $\mu$-ELASTIC factors $ms$ are unchanged. We also halve the head dimension to $d_{\mathrm{head}} = 64$, which $g = 64$ requires.

**Observation:** With $d_{\text{head}} = 64$ in both, $g = 64$ and $g = 128$ define the same model and differ only in the block size of our kernels: their mean losses differ by $0.001$ nats, and $g = 64$ takes $16\%$ more time per step than $g = 128$ in our implementation. Halving the head dimension lowers the mean loss by $0.003$ nats at $g = 128$ and $0.004$ nats at $g = 64$, about twice the run-to-run standard deviation, mostly at Low ($0.007$ and $0.009$ nats) and by at most $0.002$ nats at Max, at $2\%$ more time per step and $5\%$ more memory at $g = 128$. The head dimension thus has a small effect near run-to-run variation, and the base width none; we keep $g = d_{\text{head}} = 128$.

**(12) Optimization HPs.** We vary the gradient clipping (global norm $0.5$, $1$ and $2$), the AdamW $\epsilon$ ($10^{-6}$, $10^{-8}$ and $10^{-10}$) and the $z$-loss coefficient $\lambda_z$ ($10^{-3}$ and $10^{-2}$) around the defaults of Table 8: global norm $4$, $\epsilon = 10^{-14}$ and $\lambda_z = 10^{-4}$.

**Observation:** The defaults are within $0.002$ nats in mean loss of clipping at $0.5$ or $2$, $\epsilon = 10^{-8}$ or $10^{-10}$, and $\lambda_z = 10^{-3}$ or $10^{-2}$. Clipping at $1$ is $0.005$ nats worse, and $\epsilon = 10^{-6}$ is $0.007$ nats worse on the mean and $0.015$ nats worse at Max, so the recipe is robust to these settings except for a large $\epsilon$.