# Tevatron-Elastic: A Unified Abstraction for Training Elastic Retrievers and Rerankers

Yu Wang
yuki.wang@utah.edu
University of Utah
Salt Lake City, UT, USA

Shengyao Zhuang
s.zhuang@uq.edu.au
The University of Queensland
Brisbane, QLD, Australia

Xueguang Ma
x93ma@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Zongyu Wu
zongyuwu@psu.edu
The Pennsylvania State University
University Park, PA, USA

Jimmy Lin
jimmylin@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Vivek Srikumar
svivek@cs.utah.edu
University of Utah
Salt Lake City, UT, USA

Zhichao Xu
zhichao.xu@utah.edu
University of Utah
Salt Lake City, UT, USA

## Abstract

A single model scale challenges the flexibility of a production retrieval system: some settings need it faster, others need a smaller index, and the right trade-off changes with the workload. In the context of information retrieval (IR), a transformer-based model can be made smaller in three ways — using fewer layers, passing fewer tokens through the upper layers, or producing a shorter embedding — and each way saves a different resource. These options have been studied in isolation, each as its own method with its own code and training setup, which makes them hard to combine or adapt to a new model. In this work, we present `Tevatron-Elastic` to bring all three under one unified abstraction: a single object names any size the model can run at, and a short schedule lists the sizes to train. Training then produces one checkpoint that serves all of those sizes, and at deployment the user has the liberty to pick any of them. The same abstraction covers both retrievers and rerankers and both encoder and decoder models, as it works through interfaces that Hugging Face transformers already expose; adding a new backbone can be a configuration change rather than a model rewrite. Prior methods — Matryoshka embeddings [22], early exit [27, 37], 2D Matryoshka (Starbucks) [47], and layerwise token compression (LTC) [48] — become special cases of our unified abstraction. With `Tevatron-Elastic`, we also propose a new elastic algorithm Matryoshka LTC (MLTC) to jointly train several token-compression ratios in one retriever checkpoint. To validate our framework, we train 20 checkpoints across three backbones and two tasks: the quality curves are smooth, full-point quality remains close to that of models trained for a single size, and a controlled study confirms the wall-clock speedups. We release the framework and all checkpoints to support elastic retrieval research.

This work is licensed under a Creative Commons Attribution 4.0 International License.
*SIGIR-AP 2026, Singapore, Singapore*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3033-7/2026/12
https://doi.org/10.1145/3848000.3850194

## CCS Concepts

• **Information systems** → **Retrieval models and ranking**; **Document representation**.

## Keywords

Dense retrieval; reranking; model compression; elastic inference; Matryoshka representations.

**ACM Reference Format:**
Yu Wang, Shengyao Zhuang, Xueguang Ma, Zongyu Wu, Jimmy Lin, Vivek Srikumar, and Zhichao Xu. 2026. Tevatron-Elastic: A Unified Abstraction for Training Elastic Retrievers and Rerankers. In *Proceedings of the 2026 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region (SIGIR-AP 2026), December 13–15, 2026, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3848000.3850194

Code: `Tevatron-Elastic`

Models: Tevatron Elastic Collection

## 1 Introduction

A transformer-based information retrieval (IR) model [25, 42] has three largely separable axes that control its cost at inference time:

- **depth** — how many transformer layers are executed before a readout. Fewer layers mean less encoder/scorer compute and lower latency [6, 27, 34, 37].
- **token** — how many sequence positions flow through the upper layers. Pooling the sequence partway up the stack reduces the floating-point operations (FLOPs) of those upper layers [15, 46].
- **width** — the dimensionality of the output embedding. A smaller embedding reduces the size of the stored index and the cost of nearest-neighbor search [21, 22].

A useful mental model to group these axes is by the deployment cost each one reduces: depth and token reduce *compute*, while width reduces *storage and search*. Depth and token act at compute time and leave the index unchanged; width acts at storage time and leaves the per-input compute unchanged.

Prior work on elastic modeling tends to commit to a single axis. Matryoshka representation learning (MRL) varies width [22]; layer-dropping and early-exit methods vary depth [9, 27, 37]; Zhang et al. [46] and Zhuang et al. [48] compress the retriever and reranker on the token axis; and 2D Matryoshka retrievers such as 2DMSE and Starbucks vary depth and width together [24, 47]. Each of the aforementioned methods is effective in its own setting, but each comes as a separate implementation with its own model class definition and training objective, so combining them, or adapting one to a new task or backbone, requires re-implementation.

We posit that these compression axes are complementary rather than competing: because they reduce different costs, they can be applied independently and composed. Existing work, however, provides no single abstraction to unify the definitions of different compressed model classes, as well as the corresponding training and inference. This brings us to our central research question:

*Can we name and train these choices in a single framework, while retaining the deployment trade-offs each one offers?*

In this work, we present `Tevatron-Elastic` to provide a unified abstraction of these axes based on `Tevatron` [28]. We first abstract the action of compressing a model as an *operating point*, e.g., early exit at a certain layer means defining the early exit operating point for that layer. Based on this definition, `Tevatron-Elastic` introduces two core components. A frozen dataclass, `Granularity`, names any operating point with fields for depth, width, and token compression; and a `GranularitySchedule` lists the operating points to train the underlying elastic model. Specifically, the training loop sums a per-granularity loss over the schedule from a shared backbone forward pass and any token-specific upper-layer passes, producing one checkpoint that serves every listed operating point. Instead of editing the model internals, the abstraction routes through the hidden-states tuple, the layer module list, and the pooling step exposed by Hugging Face transformers. Depth and width need only configuration changes for a new backbone; token compression additionally needs a per-family upper-layer runner.

Our core contribution is the unified abstraction that accommodates different types of compression strategies:

(1) one abstraction (Section 3) covers retrieval and reranking across three compression axes and five families (BERT, ModernBERT, Qwen3, Llama3, and Mistral) without rewriting model classes;
(2) the same abstraction expresses prior elastic methods: MRL, early exit, 2D Matryoshka (Starbucks), and LTC as configurations, and recovers their characteristic quality–cost curves under a single shared training recipe (Section 4);
(3) the operating points trained jointly behave as expected — the quality curves from our empirical studies are smooth and full-point quality remains close to that of dedicated single-point models (Section 4). Controlled measurements show that inference speedups follow the analytic cost model (Section 4.3).
(4) a new elastic algorithm *Matryoshka Layerwise Token Compression* (MLTC), which jointly trains several token-compression ratios in one checkpoint, extending prior token compression methods [46, 48]. MLTC uses the framework's existing schedule and token readout path; supporting several mid-stack pooling points would otherwise require modifying the model class.

We do not claim that `Tevatron-Elastic` improves on prior methods in absolute quality, nor that combining all three axes is preferable to using one. In our empirical experiments, we hold the training recipe fixed (supervised fine-tuning from base checkpoints, without the pre-training some prior systems use [14, 43, 47]), so that we can study the trained elastic checkpoints in isolation. We release our framework and checkpoints to help researchers reproduce existing elastic model recipes and develop new ones.

## 2 Related Work and Background

We review IR compression and the broader model-compression literature (Section 2.1), then define the notation used throughout this paper (Section 2.2).

### 2.1 Prior methods

We organize prior work by compression axis, focusing on retrieval or ranking models that are *elastic* — adjustable in cost after training.

*Width.* Matryoshka representation learning trains an embedding whose leading coordinates remain usable on their own, so one model serves several vector sizes and, in retrieval, several index sizes [22]. The truncation is at the output, so width changes storage and nearest-neighbor search cost [21] while leaving per-input computation unchanged.

*Depth.* Early-exit and layer-dropping methods read out the representations at an intermediate layer to trade quality for fewer layers of compute, either with a fixed exit or an input-adaptive one [9, 27, 37]. In IR, the resulting encoder or reranker executes fewer transformer layers.

*Token.* Sequence-compression methods shorten the token sequence inside the stack so the upper layers process fewer positions [15]. Prior works Jasper-Token-Compression-600M [46] and layerwise token compression (LTC) [48] pool the sequence of tokens at a chosen layer and run the remaining layers on the shortened sequence for retrieval and reranking, respectively.

*Composition.* A few methods combine axes. 2D Matryoshka embeddings vary depth and width together [24], and Starbucks trains a retriever elastic on both, so one checkpoint serves a grid of (layer, dim) points [47]. These are the closest prior systems to ours; each, however, fixes one particular pair of axes and one task.

*Relation to general model compression.* The three axes above are the IR-specific face of a broader model-compression literature, which we group into two lines. The first produces a single smaller model: structured pruning removes layers, heads, or channels [36, 40], distillation trains a compact student from a larger teacher [17, 31], and quantization lowers the numerical precision of weights and activations [5, 13, 26]; in IR, distilling cross-encoders into cheaper rankers [18, 41] and efficiency-oriented learned sparse retrieval [23, 39, 43] follow this pattern. The second line trains one network that can be executed at several sizes after training: slimmable and once-for-all networks vary width and depth in convolutional models [2, 45], LayerDrop and DynaBERT do so for transformers [10, 19], MatFormer and Flextron nest sub-models inside large language models [3, 7]. Token reduction sits between the two lines: PoWER-BERT learns a fixed retention configuration

and retrains for it [15], whereas token merging applies at inference time without retraining [1]; the token axis in our framework instead trains several keep ratios into one checkpoint. `Tevatron-Elastic` belongs to the second, train-once-serve-many line. It is a domain-specific adaptation of these ideas rather than a new compression operator: the depth, token, and width axes are the counterparts of layer, sequence, and width elasticity for the two readouts that IR models use, a pooled embedding and a relevance score. Weight quantization, from the first line, acts on the stored weights and can be applied to any operating point of an elastic checkpoint, so it is complementary to the abstraction we propose; distillation likewise enters the framework as a per-granularity objective rather than as a competing method (Section 3).

*Our position.* In existing works, each axis (or fixed pair) is realized as a separate method with its own model class definition, implementation, and training strategy, and almost always on a single task. We do not introduce a new pooling operation; instead, we provide one unified abstraction in which depth, token, and width are named uniformly, the set of operating points to train is written as data, and the methods above — MRL, early exit, LTC, and 2D Matryoshka — appear as configurations for both retrieval and reranking. The per-operating-point objective is itself pluggable and can be a standard contrastive loss, a learning-to-rank loss [4, 29, 35], or distillation [17, 41] (Section 3). MLTC is one new configuration enabled by this interface: it applies the same token-pooling path to several ratios in a single retriever checkpoint.

## 2.2 Notation

We set up notation shared by the three axes, which Section 3 then unifies. A transformer with $L$ layers maps an input of $T$ tokens to a sequence of hidden states; write $H^{(\ell)} \in \mathbb{R}^{T\times D}$ for the output of layer $\ell$, with $D$ the hidden size. A dense encoder pools the top layer to a vector $z = \mathrm{pool}\big(H^{(L)}\big) \in \mathbb{R}^D$ (CLS, mean, or last token); a cross-encoder maps $H^{(L)}$ to a scalar relevance score. An operating point specifies the layer, token count, or width for a readout.

*Width (MRL).* Width keeps the first $d \le D$ coordinates of the pooled vector, $z_{1:d}$, and scores with the truncated embedding. Training sums the losses for all requested widths $\mathcal{D} = \{d_1, \dots, d_k\}$, evaluating the same objective on each nested prefix [22]:

$$\mathcal{L}_{\mathrm{MRL}} = \sum_{d\in\mathcal{D}} \mathcal{L}\big(z_{1:d}\big). \tag{1}$$

The forward pass is unchanged; only the output dimension varies, so width affects index size and search cost, not per-input compute.

*Depth (early exit).* Depth reads out at an intermediate layer $\ell \le L$ instead of the top, using $\mathrm{pool}\big(H^{(\ell)}\big)$ (or the scalar head applied at layer $\ell$). Training over a set of exit layers $\mathcal{E} = \{\ell_1, \dots, \ell_k\}$ gives

$$\mathcal{L}_{\mathrm{depth}} = \sum_{\ell\in\mathcal{E}} \mathcal{L}\big(\,\mathrm{pool}\big(H^{(\ell)}\big)\big). \tag{2}$$

At inference, an exit at layer $\ell$ executes only those $\ell$ layers, giving an approximate relative compute cost of $\ell/L$.

*Token (LTC).* Token compression pools the sequence at a chosen layer $p$, reducing its length from $T$ to $m = \max\big(1, \mathrm{round}\big(rT\big)\big)$ in our implementation for a keep-ratio $r \in (0, 1]$, then runs the remaining layers $p{+}1, \dots, L$ on the shortened sequence [48]. Writing $\mathrm{pool}_{\mathrm{seq}}$ for the sequence pooling and $f_{p:L}$ for the upper-layer stack,

$$\tilde{H}^{(p)} = \mathrm{pool}_{\mathrm{seq}}\big(H^{(p)}, r\big), \qquad z = \mathrm{pool}\big(f_{p:L}(\tilde{H}^{(p)})\big). \tag{3}$$

The layers below $p$ still run on the full sequence, so the compute saving applies only to the upper layers and is largest for long inputs.

Although the three choices intervene at different components of transformer computations, they all read from or transform the same hidden-state sequence. Naturally, this shared structure suggests a common interface, which we illustrate next in Section 3.

# 3 Framework Design

To turn this shared structure into a framework, we first clarify the concept of residual stream and the readout. We then describe the requirements for a unified elastic framework.

*Preliminary: residual stream and readout.* A backbone processes an input and returns a sequence of hidden representations as output. This sequence is referred to as the *residual stream* in the literature [8]. When we set `output_hidden_states=True`, the standard Hugging Face interface returns it as a tuple $\big(H^{(0)}, H^{(1)}, \dots, H^{(L)}\big)$, where $H^{(0)}$ is the embedding output and $H^{(\ell)} \in \mathbb{R}^{T\times D}$ is the output after layer $\ell$. A *readout* is the method we use to turn this residual stream into the model output at the operating point: choose a layer, optionally shorten its token sequence, then pool it to a retrieval vector or map it to a relevance score. For a retriever, the readout can also keep only part of the vector. Depth chooses *which* layer to read; token compression changes the sequence length $T$; and width truncates the feature dimension $D$.

```
out = backbone(**inputs, output_hidden_states=True)
H = out.hidden_states        # stream: H0, ..., HL
h = H[layer]                 # depth readout: [T, D]
z = pool(h, attention_mask)  # cls / mean / eos -> [D]
z = z[:dim]                  # width: first dim
```

Token compression is the one case that changes the stream *between* layers: it pools $H^{(p)}$ along $T$ and runs the upper layers on the shorter sequence. The remaining operations are simple choices in the readout above. This view is the bridge between the three axes.

*Desiderata.* Starting from this common readout, a unified framework should meet four practical requirements:

(1) Name every operating point in one convention, whether it changes depth, token count, or embedding width.
(2) Let users combine operating points without new modeling code, and select the points to train in a small configuration rather than in a new training loop.
(3) Work with the interfaces that transformer models already expose, so adding a backbone requires only configuration changes instead of rewriting the elastic model.
(4) Train and serve many operating points efficiently, so users can deploy one checkpoint at different sizes rather than train and maintain a separate model for each size.

The rest of this section answers these four goals in turn.

**Table 1: Schedule grammar: configurations for individual compression axes, their compositions, and prior methods.**

| what to train | spec / preset | reproduces |
|---|---|---|
| depth-only (early exit) | `"2,4,6,8,10,12"` / `pure_depth` | early exit |
| width-only (MRL) | `pure_mrl(12,(32,...,768))` | MRL |
| depth × width | `starbucks_bert` / `diagonal(...)` | Starbucks |
| token-only (reranker) | `"20@0.6"` | LTC |
| token-only (retriever) | `"28@1.0/20,...,28@0.4/20"` | MLTC (this work) |
| width + token | `"28:256@0.6/20"` | — (composition) |
| full three-axis | `"24:256@0.6/20"` | left open (Section 4) |

*One description for every operating point.* A frozen dataclass specifies the exit layer, with optional width and token settings:

```
@dataclass(frozen=True)
class Granularity:
    layer: int                       # readout / exit layer
    dim: int | None = None           # width: first dim
    keep_ratio: float | None = None  # token: fraction kept
    pool_layer: int | None = None    # token: pool here
```

A `Granularity` prints as a compact, collision-free key. Examples include `L12`, `L6xD128`, `L20xR0.8`, and `L28xR0.6xP20`. The key doubles as the operating point's name in logs and on disk. Fields are independent: setting only `layer` gives depth, adding `dim` gives width, and adding `keep_ratio` with `pool_layer` gives token compression. The reranker's legacy token path instead uses `layer` for the pooling point and reads out at the final layer; `L20xR0.6`, for example, pools at layer 20 and exits at layer 28 on Qwen3-0.6B.

*A schedule instead of custom training code.* The set of operating points to train is a list, written as a short spec string or JSON:

```
schedule = GranularitySchedule.from_spec("2:32,6:128,12:768")
```

Table 1 maps spec strings to prior methods. The grammar covers single axes and their compositions, so selecting operating points requires only configuration changes.

*How readout operates on the standard Hugging Face interface.* Because the readout only uses the hidden-states tuple, the layer module list, and a pooling step — all standard across Hugging Face transformers — we can use one function to realize all three axes for any backbone, parameterized by a `Granularity` `g`:

```
def _readout(self, hidden_states, attention_mask, g):
    if g.pool_layer is not None:   # token: pool mid-stack
        h_p = hidden_states[g.pool_layer]
        pooled_h, mask = pool_sequence(
            h_p, attention_mask, g.keep_ratio)
        h = self._range_runner(
            self.encoder, pooled_h, mask,
            g.pool_layer, end=g.layer)
    else:                          # depth: read at g.layer
        h, mask = hidden_states[g.layer], attention_mask
    reps = self._pooling(h, mask)  # cls / mean / eos
    if g.dim is not None:          # width: truncate
        reps = reps[:, :g.dim]
    return F.normalize(reps) if self.normalize else reps
```

**Table 2: Adding a backbone is a launch flag, not new model code. The pooling column is the only per-backbone choice for the depth and width axes.**

| backbone | family | pooling | new model code |
|---|---|---|---|
| BERT-base [6] | encoder | cls | — |
| ModernBERT-base [34] | encoder | cls | — |
| Qwen3-0.6B [44] | decoder (GQA) | eos | — |
| Llama3-{1B,3B,8B} [16] | decoder (GQA) | eos | — (full path verified) |
| Mistral-7B-v0.3 [20] | decoder (GQA) | eos | — (full path verified) |

The token path alone needs a per-family upper-layer runner to rebuild attention masks and decoder rotary position embeddings for the shortened sequence:

```
@register_upper_runner("qwen3", "qwen2", "mistral", "llama")
# end=None runs the upper layers to the top.
def _decoder_upper(backbone, hidden, mask, l_star,
                   end=None): ...
```

*Matryoshka Layerwise Token Compression (MLTC).* MLTC extends LTC to jointly train several token-compression ratios in one retriever checkpoint. Its `GranularitySchedule` lists the ratios, and the training loss sums their individual losses, as in MRL.

All ratios share the token readout, so MLTC adds no model path. The token runner and schedule support several mid-stack pooling points without modifying Hugging Face model classes. All five families in Table 2 support depth and width; token compression needs a per-family runner and excludes ModernBERT.

*Joint training.* Each tower runs one shared backbone forward pass per batch. We compute a readout for each operating point; token points also run the upper layers on a pooled sequence. The training loss sums the per-granularity losses:

```
# One shared forward pass per tower.
out_q = model(**query, output_hidden_states=True)
out_p = model(**passage, output_hidden_states=True)
loss = sum(
    gran_loss(out_q, out_p, g) for g in schedule.points)
```

The per-granularity loss `gran_loss` is a pluggable function of the readouts, not part of the training framework: the schedule decides *which* operating points to train, while `gran_loss` decides *how* each is supervised. In our experiments it is the task's standard objective. For the retriever it is in-batch InfoNCE [29] over the query and passage readouts, which is softmax cross-entropy over each query's candidate list together with in-batch negatives [30, 38] — equivalently, the ListNet objective [4] with a single relevant item:

```
def gran_loss(out_q, out_p, g):  # InfoNCE / top-1 ListNet
    # q: [B,d]; p: [B*n,d]; scores: [B,B*n].
    q = self._readout(out_q.hidden_states, q_mask, g)
    p = self._readout(out_p.hidden_states, p_mask, g)
    scores = (q @ p.T) / self.temperature
    target = torch.arange(B, device=q.device) * n
    return F.cross_entropy(scores, target)
```

In every case the schedule and the readout are unchanged; replacing ListNet with ListMLE or a distillation loss is a matter of plugging in a different `gran_loss`. Because `gran_loss` only sees the readouts for an operating point, substituting a different objective is a local change to this one function. For an example from the learning-to-rank literature, we use ListMLE [35]. This objective replaces the ListNet term with the Plackett–Luce likelihood of the target ordering:

```
def gran_loss(out_q, out_p, g):           # ListMLE
    q = self._readout(out_q.hidden_states, q_mask, g)
    p = self._readout(out_p.hidden_states, p_mask, g)
    scores = (q @ p.T) / self.temperature
    return listmle(scores, target_order)  # Plackett-Luce
```

Distillation instead supervises each operating point with a teacher model [17, 41], where `score_with_teacher` returns the teacher's query–passage scores for the batch [41]:

```
def gran_loss(out_q, out_p, g):  # distillation
    q = self._readout(out_q.hidden_states, q_mask, g)
    p = self._readout(out_p.hidden_states, p_mask, g)
    student = (q @ p.T) / self.temperature
    teacher = score_with_teacher(query, passage)
    return F.kl_div(
        student.log_softmax(-1), teacher.softmax(-1),
        reduction="batchmean")
```

The reranker follows the same pattern with grouped cross-entropy over each query's candidate list (the positive at index 0). The training cost of a schedule follows from the same structure. The backbone forward and backward passes are shared by all operating points; a depth or width point adds only a pooling or truncation step and one loss term, whereas a token point also re-runs the layers above its pooling layer on the shortened sequence, so it adds a fraction of a forward pass, the matching backward pass through those layers, and the activations retained for it. Measuring training time and peak memory as the schedule grows remains future work.

*Checkpoint serving.* We load the full checkpoint and prune in place to a chosen operating point. Depth pruning physically removes the upper layers, so the compute saving applies to every input; width and token are applied at readout time:

```
model = ElasticDenseModel.load(ckpt)  # or ElasticReranker
model.prune_to(Granularity(16))       # select a point
```

We store a single checkpoint to serve all trained operating points.

*A cost model for the three axes.* A linear approximation maps a `Granularity` to its relative encode/score FLOPs: $\ell/L$ for a depth point at layer $\ell$, and $\big(p+(L-p)\,r\big)/L$ for a token point that pools at layer $p$ with keep-ratio $r$. The token estimate assumes cost linear in sequence length and ignores pooling overhead. Relative index size is $d/d_{\max}$ for width $d$. Section 4.3 checks the measured wall-clock speedup and index size reduction.

# 4 Experiments

This section presents the experimental setup and results, examining whether the proposed framework meets the desiderata above.

## 4.1 Experimental setup

*Backbones.* We experiment with three backbone models: BERT-base [6] (110M, encoder, CLS pooling), ModernBERT-base [34] (150M, encoder, CLS pooling, alternating global/local attention), and Qwen3-0.6B [44] (decoder, GQA, last-token pooling). Depth and width are evaluated on all three; token experiments use Qwen3 only. Llama3 family models [16] and Mistral-7B [20] are used only to verify the backbone-agnostic path end to end.

*Data and evaluation.* All models are trained on RLHN-680K [33] with `query:` and `passage:` prefixes, at input length 512, despite using different tokenizers. We evaluate on BEIR-15 [32] as a single macro average (cqadupstack collapsed to the mean of its 12 subforums and counted once; MS MARCO dev included as one member), reporting nDCG@10. Retrieval reports absolute nDCG@10; reranking reports nDCG@10 after reranking BM25 top-100, so the BM25 first stage (0.415) is the reference. The per-backbone *plain* (non-elastic, single full point) model is the baseline against which we read each compression curve.

*What the experiments assess.* The experiments assess the desiderata discussed in Section 3. The backbone coverage exercises the common interface across encoder and decoder families. The quality curves test whether a common operating-point description and schedule can express the expected depth, width, token, and composed trade-offs. The experiments also measure the cost of joint training at the full point (the *elasticity tax*: elastic full-point minus the dedicated plain full-point), and separately measure the deployment costs that each axis is intended to change. We focus less on absolute performance numbers and more on diagnosing the framework. Every number we report comes from a single training run per configuration; we do not repeat runs with different seeds and so cannot report variance. We accordingly draw conclusions from the shape of each curve, which is consistent across the three backbones for the depth, width, and depth×width axes (the token axis is evaluated on Qwen3-0.6B only), rather than from individual small differences, and we flag below the comparisons whose margins may lie within run-to-run variation, which we do not measure.

## 4.2 Results and analysis

We first evaluate the single-axis operating points and then their compositions. These configurations correspond to prior elastic methods where applicable. Figures 1 to 3 show the quality curves (BEIR-15 nDCG@10) while Table 3 reports complete results for the Qwen3-0.6B model. We skip the detailed BERT and ModernBERT results for the sake of space. Dataset abbreviations: ARG arguana, CFE climate-fever, DBP dbpedia-entity, FEV fever, FIQ fiqa, HOP hotpotqa, NFC nfcorpus, NQ nq, QUO quora, SCD scidocs, SCF scifact, TRC trec-covid, TOU webis-touche2020, MSM MS MARCO dev, CQA cqadupstack (mean of 12). Granularity labels: $\ell$:$d$ exit layer $\ell$ and dim $d$ (retriever); L$\ell$ exit layer $\ell$ (reranker); L$\ell$×R$r$×P$p$ exit layer $\ell$, keep-ratio $r$, pool layer $p$ (token / MLTC). For the reranker's legacy token keys L$p$×R$r$, $p$ is the pooling layer, and the exit is the final layer (28 here).

*Depth.* For the retriever, performance improves smoothly with the number of layers on all three backbones (Figure 1a; Qwen3 goes from 0.431 at layer 6 to 0.518 at layer 28, full width). The reranker

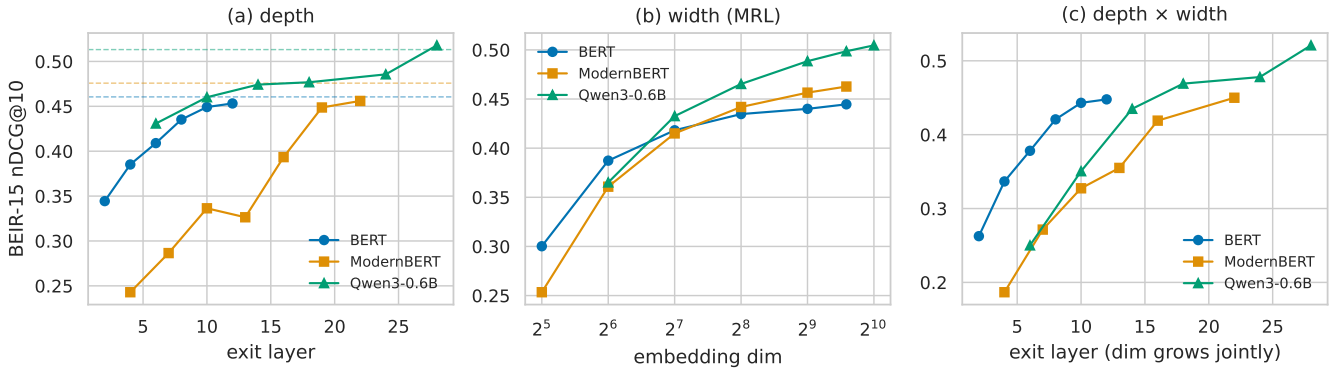


**Figure 1: Retriever quality (BEIR-15 nDCG@10) along each axis, three backbones. Dashed lines mark each backbone's plain full point. (a) depth at full width; (b) width (MRL) at full depth, log dimension axis; (c) depth×width (Starbucks), where layer and dimension grow jointly. Every curve is smooth except for ModernBERT's step near layer 13 in (a), and the elastic full points sit close to the plain reference.**

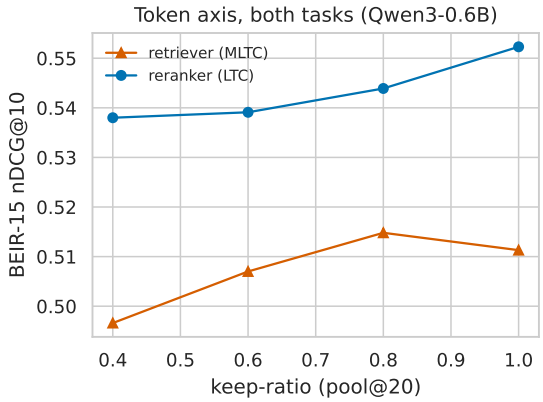


**Figure 2: Token axis on both tasks (Qwen3-0.6B): the retriever (MLTC) and reranker (LTC) curves have similar gentle shapes under the same pooling primitive.**

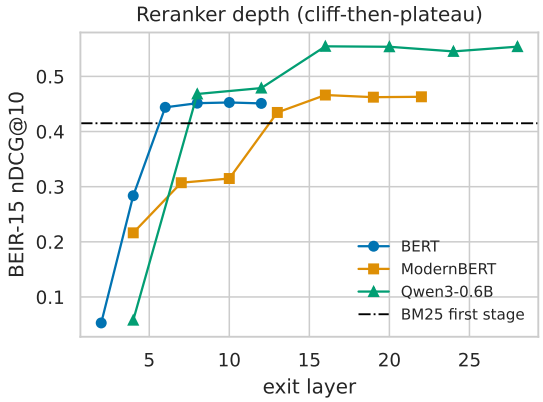


**Figure 3: Reranker depth, three backbones. A floor at the shallow end then a long plateau, distinct from the gradual retriever depth curve in Figure 1a.**

has a different curve (Figure 3): quality is very low when only a few layers run, then rises and levels off at about half the layers (Qwen3 reaches 0.554 by layer 16, matching layer 28; BERT levels off at layer 6, ModernBERT at layer 16). The retriever, which reads out a pooled vector, degrades gradually; the reranker, which reads out a single score, has a low-quality region at shallow depth and a plateau at greater depth. On both tasks, ModernBERT shows a step near layer 13 rather than a gradual rise.

*Width (MRL).* The pure-width curve is smooth and monotonic on all three backbones (Figure 1b; e.g., Qwen3 0.365 → 0.505 from dim 64 to 1024 at full depth). This reproduces MRL as a configuration; the corresponding cost is index size rather than compute (Section 4.3).

*Token (LTC Reranker and MLTC Retriever).* On the reranker, we follow Zhuang et al. [48] to pool the sequence at layer 20 and sweep the keep-ratio for multiple runs; quality falls gently and never collapses (Figure 2; Qwen3 0.552 at ratio 1.0 to 0.538 at ratio 0.4). We then apply the same pooling operation to the *retriever* — the identical code, now on a bi-encoder — and obtain a curve with a similarly gentle shape (0.497 to 0.511). Different from the LTC reranker which trains a single (layer, ratio) point, we use MLTC, which jointly trains several ratios in one checkpoint. At keep-ratio 1.0, the MLTC retriever's full-point quality is close to that of the plain Qwen3-0.6B retriever (0.511 vs 0.513). We do not compare it directly with separately trained single-ratio LTC retrievers, so this experiment does not quantify the benefit of joint training on the token axis, the token-axis counterpart of the elasticity tax in Table 4; we leave this controlled comparison to future work.

*Composition: depth × width.* The 2D Matryoshka schedule recovers the Starbucks-style frontier on all three backbones (Figure 1c); each frontier is smooth and monotonic (BERT 0.263 → 0.448; Qwen3 0.250 → 0.521; ModernBERT 0.187 → 0.450 across six operating points from one checkpoint). The shape matches Starbucks. This is a comparison of curve shape rather than of numbers.

**Table 3: Complete per-dataset BEIR-15 results (nDCG@10) for Qwen3-0.6B retrievers and rerankers.**

| Model | ARG | CFE | DBP | FEV | FIQ | HOP | NFC | NQ | QUO | SCD | SCF | TRC | TOU | MSM | CQA | **AVG** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Qwen3-0.6B retriever, plain.* | | | | | | | | | | | | | | | | |
| 28:1024 | 0.650 | 0.344 | 0.386 | 0.855 | 0.393 | 0.738 | 0.344 | 0.539 | 0.858 | 0.225 | 0.686 | 0.704 | 0.234 | 0.392 | 0.349 | **0.513** |
| *Qwen3-0.6B retriever, depth.* | | | | | | | | | | | | | | | | |
| 6:1024 | 0.488 | 0.260 | 0.314 | 0.804 | 0.248 | 0.603 | 0.279 | 0.405 | 0.835 | 0.168 | 0.580 | 0.646 | 0.219 | 0.330 | 0.284 | **0.431** |
| 10:1024 | 0.554 | 0.292 | 0.330 | 0.811 | 0.289 | 0.609 | 0.314 | 0.447 | 0.844 | 0.186 | 0.605 | 0.739 | 0.257 | 0.341 | 0.283 | **0.460** |
| 14:1024 | 0.592 | 0.321 | 0.326 | 0.817 | 0.320 | 0.621 | 0.317 | 0.493 | 0.845 | 0.196 | 0.620 | 0.767 | 0.237 | 0.350 | 0.289 | **0.474** |
| 18:1024 | 0.634 | 0.312 | 0.338 | 0.820 | 0.336 | 0.619 | 0.320 | 0.492 | 0.838 | 0.199 | 0.616 | 0.753 | 0.210 | 0.355 | 0.310 | **0.477** |
| 24:1024 | 0.641 | 0.293 | 0.368 | 0.837 | 0.353 | 0.673 | 0.312 | 0.509 | 0.845 | 0.203 | 0.639 | 0.699 | 0.230 | 0.368 | 0.314 | **0.486** |
| 28:1024 | 0.661 | 0.339 | 0.389 | 0.856 | 0.387 | 0.734 | 0.346 | 0.550 | 0.854 | 0.214 | 0.683 | 0.735 | 0.268 | 0.393 | 0.359 | **0.518** |
| *Qwen3-0.6B retriever, width (MRL).* | | | | | | | | | | | | | | | | |
| 28:64 | 0.470 | 0.231 | 0.213 | 0.666 | 0.265 | 0.392 | 0.226 | 0.334 | 0.820 | 0.172 | 0.481 | 0.482 | 0.228 | 0.311 | 0.183 | **0.365** |
| 28:128 | 0.542 | 0.278 | 0.295 | 0.786 | 0.318 | 0.567 | 0.279 | 0.418 | 0.838 | 0.197 | 0.549 | 0.577 | 0.244 | 0.351 | 0.251 | **0.433** |
| 28:256 | 0.579 | 0.301 | 0.338 | 0.816 | 0.351 | 0.644 | 0.306 | 0.460 | 0.846 | 0.207 | 0.597 | 0.625 | 0.257 | 0.368 | 0.287 | **0.465** |
| 28:512 | 0.612 | 0.320 | 0.362 | 0.833 | 0.375 | 0.689 | 0.323 | 0.487 | 0.853 | 0.215 | 0.620 | 0.663 | 0.275 | 0.381 | 0.320 | **0.489** |
| 28:768 | 0.623 | 0.333 | 0.370 | 0.846 | 0.384 | 0.713 | 0.333 | 0.496 | 0.855 | 0.220 | 0.640 | 0.671 | 0.271 | 0.387 | 0.338 | **0.499** |
| 28:1024 | 0.628 | 0.343 | 0.382 | 0.855 | 0.389 | 0.726 | 0.339 | 0.503 | 0.857 | 0.222 | 0.646 | 0.676 | 0.266 | 0.391 | 0.345 | **0.505** |
| *Qwen3-0.6B retriever, depth×width (2D Matryoshka).* | | | | | | | | | | | | | | | | |
| 6:64 | 0.346 | 0.117 | 0.101 | 0.501 | 0.098 | 0.228 | 0.139 | 0.144 | 0.775 | 0.106 | 0.318 | 0.457 | 0.195 | 0.141 | 0.085 | **0.250** |
| 10:128 | 0.460 | 0.216 | 0.195 | 0.683 | 0.176 | 0.393 | 0.210 | 0.289 | 0.824 | 0.150 | 0.453 | 0.563 | 0.248 | 0.235 | 0.160 | **0.350** |
| 14:256 | 0.565 | 0.301 | 0.287 | 0.777 | 0.272 | 0.527 | 0.291 | 0.440 | 0.842 | 0.191 | 0.564 | 0.669 | 0.246 | 0.317 | 0.237 | **0.435** |
| 18:512 | 0.620 | 0.329 | 0.326 | 0.807 | 0.333 | 0.581 | 0.313 | 0.495 | 0.839 | 0.197 | 0.601 | 0.715 | 0.242 | 0.346 | 0.293 | **0.469** |
| 24:768 | 0.657 | 0.320 | 0.351 | 0.820 | 0.351 | 0.641 | 0.302 | 0.511 | 0.842 | 0.202 | 0.638 | 0.635 | 0.226 | 0.362 | 0.312 | **0.478** |
| 28:1024 | 0.677 | 0.362 | 0.389 | 0.850 | 0.396 | 0.731 | 0.341 | 0.560 | 0.857 | 0.220 | 0.693 | 0.730 | 0.256 | 0.390 | 0.360 | **0.521** |
| *Qwen3-0.6B retriever, token (MLTC, pool@20).* | | | | | | | | | | | | | | | | |
| L28xR0.4xP20 | 0.646 | 0.349 | 0.357 | 0.827 | 0.342 | 0.685 | 0.334 | 0.532 | 0.841 | 0.211 | 0.677 | 0.712 | 0.232 | 0.370 | 0.333 | **0.497** |
| L28xR0.6xP20 | 0.657 | 0.363 | 0.369 | 0.832 | 0.366 | 0.704 | 0.329 | 0.540 | 0.850 | 0.215 | 0.683 | 0.734 | 0.245 | 0.374 | 0.342 | **0.507** |
| L28xR0.8xP20 | 0.660 | 0.369 | 0.376 | 0.845 | 0.376 | 0.713 | 0.338 | 0.546 | 0.852 | 0.217 | 0.681 | 0.765 | 0.258 | 0.378 | 0.348 | **0.515** |
| L28xR1.0xP20 | 0.655 | 0.377 | 0.381 | 0.842 | 0.368 | 0.722 | 0.339 | 0.546 | 0.855 | 0.219 | 0.669 | 0.712 | 0.245 | 0.389 | 0.349 | **0.511** |
| *Qwen3-0.6B reranker, plain.* | | | | | | | | | | | | | | | | |
| L28 | 0.604 | 0.342 | 0.449 | 0.894 | 0.432 | 0.799 | 0.392 | 0.596 | 0.853 | 0.207 | 0.762 | 0.867 | 0.357 | 0.399 | 0.390 | **0.556** |
| *Qwen3-0.6B reranker, depth.* | | | | | | | | | | | | | | | | |
| L4 | 0.011 | 0.047 | 0.062 | 0.111 | 0.009 | 0.027 | 0.130 | 0.016 | 0.006 | 0.011 | 0.011 | 0.307 | 0.102 | 0.015 | 0.005 | **0.058** |
| L8 | 0.525 | 0.251 | 0.354 | 0.848 | 0.288 | 0.724 | 0.323 | 0.403 | 0.848 | 0.173 | 0.676 | 0.721 | 0.267 | 0.287 | 0.334 | **0.468** |
| L12 | 0.535 | 0.273 | 0.365 | 0.854 | 0.295 | 0.737 | 0.329 | 0.438 | 0.856 | 0.166 | 0.674 | 0.744 | 0.290 | 0.312 | 0.314 | **0.479** |
| L16 | 0.593 | 0.340 | 0.456 | 0.890 | 0.424 | 0.796 | 0.382 | 0.591 | 0.860 | 0.201 | 0.758 | 0.870 | 0.361 | 0.399 | 0.396 | **0.554** |
| L20 | 0.600 | 0.344 | 0.457 | 0.892 | 0.420 | 0.796 | 0.390 | 0.592 | 0.829 | 0.203 | 0.759 | 0.864 | 0.368 | 0.400 | 0.391 | **0.554** |
| L24 | 0.597 | 0.330 | 0.447 | 0.889 | 0.420 | 0.793 | 0.390 | 0.589 | 0.778 | 0.199 | 0.754 | 0.858 | 0.350 | 0.400 | 0.386 | **0.545** |
| L28 | 0.603 | 0.337 | 0.453 | 0.891 | 0.422 | 0.797 | 0.389 | 0.592 | 0.848 | 0.201 | 0.755 | 0.862 | 0.368 | 0.400 | 0.389 | **0.554** |
| *Qwen3-0.6B reranker, token (LTC; pool@20, exit@28; legacy keys).* | | | | | | | | | | | | | | | | |
| L20xR0.4 | 0.572 | 0.337 | 0.450 | 0.884 | 0.406 | 0.796 | 0.380 | 0.593 | 0.755 | 0.203 | 0.755 | 0.863 | 0.305 | 0.401 | 0.368 | **0.538** |
| L20xR0.6 | 0.593 | 0.338 | 0.443 | 0.885 | 0.406 | 0.796 | 0.383 | 0.590 | 0.758 | 0.204 | 0.742 | 0.853 | 0.302 | 0.400 | 0.394 | **0.539** |
| L20xR0.8 | 0.578 | 0.338 | 0.448 | 0.886 | 0.409 | 0.796 | 0.384 | 0.591 | 0.814 | 0.203 | 0.753 | 0.859 | 0.319 | 0.401 | 0.382 | **0.544** |
| L20xR1.0 | 0.610 | 0.331 | 0.449 | 0.891 | 0.423 | 0.798 | 0.388 | 0.591 | 0.842 | 0.204 | 0.754 | 0.869 | 0.341 | 0.401 | 0.394 | **0.552** |

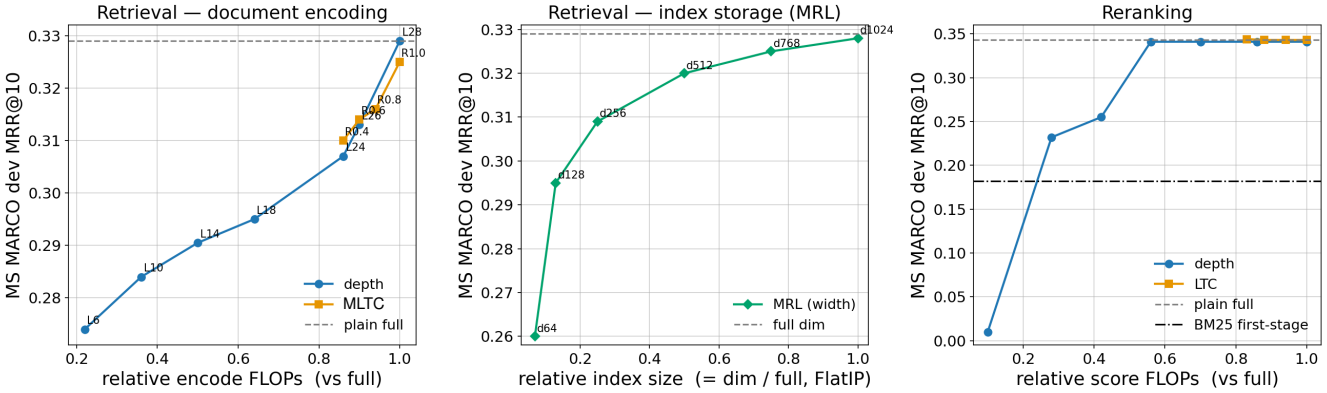


**Figure 4: Quality versus relative analytic cost. Left: retrieval document encoding, depth and MLTC, against relative FLOPs; MRL is absent because it does not change FLOPs. Center: retrieval against relative index size; only MRL moves along this axis. Right: reranking against relative score FLOPs. Depth and token reduce compute and leave the index unchanged; width reduces the index and leaves compute unchanged.**

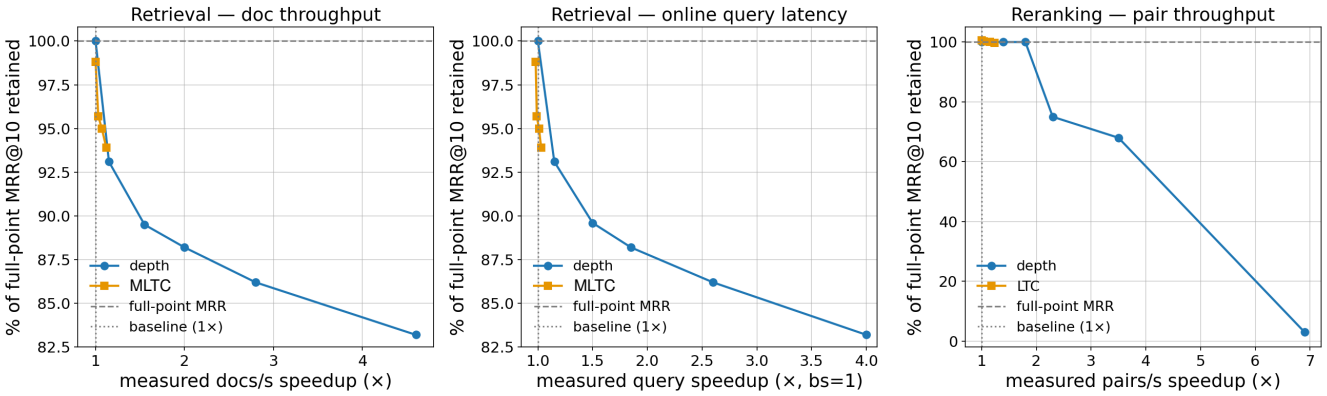


**Figure 5: Quality retained versus measured speedup. Left: retrieval document throughput. Center: online query latency at batch 1. Right: reranking throughput. Depth speeds up every path; the token axis helps document encoding modestly and online query encoding not at all, because these short, padded queries offer little compute to save and the lower layers still run.**

The released Starbucks checkpoints are trained with masked-autoencoding pretraining and on different data, neither of which is part of our recipe, and the same holds for released MRL and LTC checkpoints. Because the recipes differ, comparing against those checkpoints on BEIR-15 would measure the training recipe as much as the abstraction, which is why our plain single-point baselines, trained with the identical recipe, serve as the controlled reference at the full operating point (Table 4). The released-checkpoint evaluation, and a comparison that also controls training data at compressed operating points, are left to future work.

*Full-point elasticity tax.* Training elastic models may degrade model performance compared to standard training; we refer to this cost as the *elasticity tax*. Table 4 compares the full model trained in the elastic way to a model trained for that size alone, using the depth×width schedule so the comparison is the same across backbones. We note the measured difference is small: on the retriever it is −0.013 and −0.026 for the two encoders and +0.008 for the decoder. The full point changes modestly on the two encoder backbones, and the Qwen3 decoder has the smallest tax. Each entry is a single run, so the sign of the smallest differences, such as the +0.008 for Qwen3, should not be over-interpreted; the reading we take from these runs is that the tax is small on every backbone.

**Table 4: Elasticity tax at the full operating point (BEIR-15 nDCG@10). The depth×width schedule is used for all backbones for comparability.**

| | plain | elastic (2D) | tax |
|---|---|---|---|
| BERT @ 12:768 | 0.461 | 0.448 | −0.013 |
| ModernBERT @ 22:768 | 0.476 | 0.450 | −0.026 |
| Qwen3 @ 28:1024 | 0.513 | 0.521 | +0.008 |

*Which axes each task admits.* A retriever produces a vector and admits all three axes; a reranker produces a scalar score and admits depth and token but not width, since there is no embedding to truncate. This is the one structural exclusion in the task-by-axis matrix, explaining the reranker's smaller configuration set.

*The three-axis composition is left open.* A depth × width × token retriever is expressible today (e.g., `"24:256@0.6/20"`), and the machinery runs it (training and inference both route through the `pool_layer` branch of Section 3). We do not report an operating point for it: this paper verifies only that the machinery runs, and the quality–cost trade-off of the three-axis composition is not evaluated here; we leave this experiment to future work.

## 4.3 Efficiency and deployment trade-offs

We measure deployment costs for Qwen3-0.6B on MS MARCO dev, using MRR@10 for quality. A single in-domain corpus avoids BEIR's domain and length confounds. Timing uses one GPU and a fixed workload, separately from parallel evaluation. We report document encoding (offline, batch 128, length 512, documents/second) and query encoding (request path, batch 1, latency/query). Queries are padded to 32 tokens to cover long inputs, although they average about 8 tokens. For reranking, we host the model through a Hugging Face backend, send HTTP requests in batches of 128 query–document pairs, and compute throughput from the end-to-end request latency. Figures 4 and 5 show the measured costs and quality trade-offs on both tasks.

*The measured speedup follows the cost model.* For reranking at batch size 128, the measured speedup agrees with the prediction (1/relative FLOPs) to within about two percent at every depth (layer 4 measures 6.86× against a predicted 7.00×; layer 16, 1.75× against 1.75×). Document encoding tracks just as closely (layer 6, 4.61× measured against 4.67× predicted). When the batch is large the work is dominated by arithmetic, so counting FLOPs is enough to predict the wall-clock speedup.

*Depth and token help different paths.* At layer 16, the reranker retains the full-depth quality (0.342 MRR@10) while running 1.75× faster. The token axis gives a smaller speedup because the layers below the pooling point still run on the full sequence (pooling at layer 20 of 28 leaves about 0.71 of the work in place), though its quality falls more gently. The gap is starker for queries: cutting depth still speeds up query encoding by about 4× at layer 6, while token pooling gives no speedup (around 0.99×, slightly slower), since these short queries, padded to 32 tokens in the timing study, offer little work to save and the lower layers run regardless. Token compression is therefore a lever for long documents, while depth speeds up both the offline corpus pass and the online query. At batch 1 the query pass spends much of its time on overhead rather than arithmetic, so its speedup falls short of the FLOP prediction — which is why we measure query latency directly instead of reading it off the FLOP count for these short inputs.

*Width reduces storage and search.* With an exact inner-product index, both the stored index size and the search FLOPs are proportional to the embedding dimension, so index size measures the width axis's cost independently of hardware. Reducing the Qwen3 embedding from dim 1024 to dim 64 reduces the MS MARCO index from about 36 GB to about 2.3 GB while MRR@10 moves from 0.329 to 0.261; depth and token leave the index size unchanged.

*Depth and token save compute; width saves storage.* Taken together, the three axes address different deployment constraints: depth reduces compute on every path and is the most broadly applicable axis; the token axis reduces compute on long inputs specifically (corpus encoding, not short queries); and width reduces index storage and search cost. The cost model and the figures above indicate which axis to reach for given a latency or storage budget.

These deployment measurements concern inference rather than training overhead. Depth and width points reuse the backbone pass, whereas each token point adds an upper-layer pass on its pooled sequence. Serving several operating points from one checkpoint therefore does not imply equal training costs for every schedule; timing and memory remain separate evaluation questions.

# 5 Conclusion and Future Work

Existing model compression work in the IR community mostly targets a single compression axis at a time. We challenge this *status quo* by first providing a taxonomy of three compression axes, then proposing `Tevatron-Elastic` to study the three axes in a unified interface. Through one small dataclass and a declarative schedule, `Tevatron-Elastic` expresses depth and token compression for both tasks and width compression for retrieval. It expresses four prior methods — MRL, early exit, 2D Matryoshka, and LTC — as configurations, and runs across five encoder and decoder backbone families without rewriting their model classes (the token axis adds one small per-family shim). Across 20 checkpoints, we show that the elastic curves are smooth with few exceptions, full-point quality remains close to that of dedicated single-point models, and the measured inference speedup follows the analytic cost model. The same interface enables a new algorithm MLTC, which serves several token-compression ratios from one retriever checkpoint. We release the code and all checkpoints for community use.

Our experience building `Tevatron-Elastic` suggests several directions for extending the framework and evaluating its trade-offs. First, the token axis currently targets global-attention backbones; extending it to sliding-window-attention models (such as ModernBERT) requires a pooling rule compatible with a fixed local window, after which the token axis applies to that family as well. Second, the abstraction already expresses the full depth×width×token composition on the retriever (Section 4); a practitioner with a specific latency and storage budget can train and select that operating point directly, although we have not evaluated this composition and its quality–cost frontier remains to be measured. Third, the empirical study has some limits that the released code makes straightforward to close: for example, MLTC is not compared against separately trained single-ratio LTC retrievers; and training time and peak memory as a function of schedule size are not reported. Because a new axis, backbone, or per-granularity loss enters through a single interface, we expect further compression methods to slot in as additional configurations rather than as new systems. Finally, elastic modeling for learned sparse retrieval [11, 12, 39, 43] remains underexplored.

## References


[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2023. Token Merging: Your ViT But Faster. In *International Conference on Learning Representations (ICLR)*.

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize It for Efficient Deployment. In *International Conference on Learning Representations (ICLR)*.

[3] Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. 2024. Flextron: Many-in-One Flexible Large Language Model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

[4] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning*. 129–136.

[5] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *Advances in Neural Information Processing Systems* 35 (2022), 30318–30332.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. doi:10.18653/v1/N19-1423

[7] Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. 2023. MatFormer: Nested Transformer for Elastic Inference. arXiv:2310.07707 [cs.LG] https://arxiv.org/abs/2310.07707

[8] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A Mathematical Framework for Transformer Circuits. *Transformer Circuits Thread* (2021). https://transformer-circuits.pub/2021/framework/index.html.

[9] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 12622–12642. doi:10.18653/v1/2024.acl-long.681

[10] Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing Transformer Depth on Demand with Structured Dropout. In *International Conference on Learning Representations (ICLR)*.

[11] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2353–2359. doi:10.1145/3477495.3531857

[12] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, USA, 2288–2292. https://doi.org/10.1145/3404835.3463098

[13] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=tcbBPnfwxS

[14] Luyu Gao and Jamie Callan. 2022. Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 2843–2853. doi:10.18653/v1/2022.acl-long.203

[15] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 3690–3699. https://proceedings.mlr.press/v119/goyal20a.html

[16] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML] https://arxiv.org/abs/1503.02531

[18] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. arXiv:2010.02666 [cs.IR] https://arxiv.org/abs/2010.02666

[19] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. DynaBERT: Dynamic BERT with Adaptive Width and Depth. In *Advances in Neural Information Processing Systems*, Vol. 33.

[20] Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-Scale Similarity Search with GPUs. arXiv:1702.08734 [cs.CV] https://arxiv.org/abs/1702.08734

[22] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2022. Matryoshka Representation Learning. *Advances in Neural Information Processing Systems* 35 (2022), 30233–30249.

[23] Carlos Lassance and Stéphane Clinchant. 2022. An Efficiency Study for SPLADE Models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2220–2226. doi:10.1145/3477495.3531833

[24] Xianming Li, Zongxi Li, Jing Li, Haoran Xie, and Qing Li. 2024. 2D Matryoshka Sentence Embeddings. arXiv:2402.14776 [cs.CL] https://arxiv.org/abs/2402.14776

[25] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2022. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Springer Nature.

[26] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 87–100. https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[27] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: A Self-Distilling BERT with Adaptive Inference Time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 6035–6044. doi:10.18653/v1/2020.acl-main.537

[28] Xueguang Ma, Luyu Gao, Shengyao Zhuang, Jiaqi Samantha Zhan, Jamie Callan, and Jimmy Lin. 2025. Tevatron 2.0: Unified Document Retrieval Toolkit across Scale, Language, and Modality. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 4061–4065. doi:10.1145/3726302.3730135

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. arXiv:1807.03748 [cs.LG] https://arxiv.org/abs/1807.03748

[30] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 5835–5847. doi:10.18653/v1/2021.naacl-main.466

[31] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. arXiv:1910.01108 [cs.CL] https://arxiv.org/abs/1910.01108

[32] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[33] Nandan Thakur, Crystina Zhang, Xueguang Ma, and Jimmy Lin. 2025. Hard Negatives, Hard Lessons: Revisiting Training Data Quality for Robust Information Retrieval with LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 9064–9083. doi:10.18653/v1/2025.findings-emnlp.481

[34] Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. arXiv:2412.13663 [cs.CL] https://arxiv.org/abs/2412.13663

[35] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning* (Helsinki, Finland) *(ICML '08)*. Association for Computing Machinery, New York, NY, USA, 1192–1199. doi:10.1145/1390156.1390306

[36] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. In *International Conference on Learning Representations (ICLR)*.

[37] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 2246–2251. doi:10.18653/v1/2020.acl-main.204

[38] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations*. https://openreview.net/forum?id=zeFrfgyZln

[39] Zhichao Xu, Aosong Feng, Yijun Tian, Haibo Ding, and Lin Lee Cheong. 2025. CSPLADE: Learned Sparse Retrieval with Causal Language Models. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 99–114. doi:10.18653/v1/2025.ijcnlp-long.7

[40] Zhichao Xu, Ashim Gupta, Tao Li, Oliver Bentham, and Vivek Srikumar. 2024. Beyond Perplexity: Multi-dimensional Safety Evaluation of LLM Compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 15359–15396. doi:10.18653/v1/2024.findings-emnlp.901

[41] Zhichao Xu, Zhiqi Huang, Shengyao Zhuang, and Vivek Srikumar. 2025. Distillation versus Contrastive Learning: How to Train Your Rerankers. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (Eds.). The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, Mumbai, India, 564–578. doi:10.18653/v1/2025.findings-ijcnlp.33

[42] Zhichao Xu, Fengran Mo, Zhiqi Huang, Crystina Zhang, Puxuan Yu, Bei Wang Phillips, Jimmy Lin, and Vivek Srikumar. 2026. A Survey of Model Architectures in Information Retrieval. *Transactions on Machine Learning Research* (2026). Survey Certification. https://openreview.net/forum?id=xAIbTbHRrX

[43] Zhichao Xu, Shengyao Zhuang, Crystina Zhang, Xueguang Ma, Yijun Tian, Maitrey Mehta, Jimmy Lin, and Vivek Srikumar. 2026. LACONIC: Dense-Level Effectiveness for Scalable Sparse Retrieval via a Two-Phase Training Curriculum. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Australia) *(SIGIR '26)*. Association for Computing Machinery, New York, NY, USA, 4298–4304. doi:10.1145/3805712.3809869

[44] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[45] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[46] Dun Zhang, Ziyang Zeng, Yudong Zhou, and Shuyang Lu. 2025. Jasper-Token-Compression-600M Technical Report. arXiv:2511.14405 [cs.IR] https://arxiv.org/abs/2511.14405

[47] Shengyao Zhuang, Shuai Wang, Fabio Zheng, Bevan Koopman, and Guido Zuccon. 2026. Starbucks: Improved Training for 2D Matryoshka Embeddings. In *Advances in Information Retrieval: 48th European Conference on Information Retrieval, ECIR 2026, Delft, The Netherlands, March 29 – April 2, 2026, Proceedings, Part I* (Delft, The Netherlands). Springer-Verlag, Berlin, Heidelberg, 67–82. doi:10.1007/978-3-032-21289-4_5

[48] Shengyao Zhuang, Zhichao Xu, and Ivano Lauriola. 2026. Layer-wise Token Compression for Efficient Document Reranking. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Australia) *(SIGIR '26)*. Association for Computing Machinery, New York, NY, USA, 4426–4432. doi:10.1145/3805712.3809871