REPRESENTAX: SCALABLE REPRESENTATION LEARNING IN JAX

Anonymous authors
Paper under double-blind review

ABSTRACT

Learned representations shape how models organize information and generalize beyond their training examples. As learning systems evolve to incorporate information from increasingly diverse modalities, a flexible and efficient foundation for representation-learning research becomes essential to enable progress. We introduce Representax, an open-source JAX framework for developing and training representation-learning models across modalities. Representax unifies extensible training and evaluation tasks with established methods spanning dense and late-interaction retrieval, multimodal contrastive learning, reward modeling, and joint-embedding predictive learning. Its shared training infrastructure supports model adaptation, multimodal data processing, memory-efficient optimization, and distributed execution on GPUs and TPUs. To demonstrate its capabilities, we compare training performance against established frameworks, train and evaluate retrieval and multimodal models, and measure multi-GPU strong scaling. These experiments demonstrate competitive throughput, improvements on held-out retrieval tasks, and efficient scaling. Representax provides a foundation for investigating how representations are learned and adapted, and for taking new ideas from exploration to scale.

1 INTRODUCTION

A recurring lesson in artificial intelligence is the power of general methods that scale with computation over approaches built around hand-engineered knowledge (Sutton, 2019). In representation learning, the emphasis shifts from specifying representations by hand to developing methods that learn them from observations of the world (Bengio et al., 2013). The research challenge is to learn representations that both capture meaningful structure in the world and support prediction, generation, and action. This development spans early hierarchies of visual features and subsequent reconstruction-based and predictive approaches to learning compact encodings, linguistic relationships, and speech features (Fukushima, 1980; Hinton & Salakhutdinov, 2006; Mikolov et al., 2013; Baevski et al., 2020). Model architectures and learning signals shape not only what representations capture, but how efficiently they are learned and how well they transfer to new settings. In World Models, compressed visual observations and learned temporal dynamics support a latent simulation in which an agent can learn a policy before transferring it to the original environment (Ha & Schmidhuber, 2018). Latent diffusion makes image synthesis more computationally tractable by operating in a learned latent space (Rombach et al., 2022). In these systems, representations provide more than features for a downstream task: they define the spaces in which simulation, policy learning, and generation take place.

Multimodal learning extends this pursuit across different forms of observation. CLIP learns image-text alignment jointly through contrastive supervision (Radford et al., 2021), while systems such as BLIP-2 connect separately pretrained visual and language components (Li et al., 2023b). We suspect that future learning systems will be natively multimodal throughout pretraining and post-training, integrating information from a variety of data modalities. A longer-term ambition is to learn unified representations of the world from these different observations. Huh et al. (2024) demonstrate increasing alignment between visual and language representations as models scale, and propose the Platonic Representation Hypothesis: that this convergence reflects progress towards a shared statistical model of reality. Research in this direction examines how representations are learned, aligned,

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.

and adapted, and when shared structure supports transfer across tasks and modalities, prediction, and decision-making.

Progress towards this natively multimodal future requires software that enables researchers to develop and test new ideas. Research in representation learning spans several interdependent choices: model architectures, mechanisms for combining modalities, learning objectives, relationships used as supervision, and training recipes. Implementing a new idea should require changes that express the method itself, with minimal additional code to integrate it into a complete experiment. A framework for this research must support flexible experimentation, efficient execution, and consistent evaluation against relevant benchmarks and reference implementations.

We introduce Representax, a JAX framework for representation-learning research towards natively multimodal models. The library is built on unified abstractions for models, learning tasks, data, and evaluation. Together, they connect method development to a shared training system while keeping scientific decisions separate from execution strategies. Within supported configurations, researchers can optimize execution for the available hardware without coupling their model and objective definitions to a particular device topology.

Representax extends the JAX ecosystem (Bradbury et al., 2018), building on its composable transformations and compiled GPU/TPU execution. It integrates Equinox for native models (Kidger & Garcia, 2021), Grain for data processing (Ritter et al., 2023), Optax for optimization (DeepMind et al., 2020), and Orbax for checkpointing (Gaffney et al., 2026). For our initial implementation, we select established methods that provide meaningful benchmarks and building blocks for further research: dense and late-interaction retrieval, supervised similarity and ranking, reward modeling, multimodal contrastive learning, masked language modeling, and joint-embedding prediction.

We evaluate Representax through matched comparisons with established implementations, end-to-end adaptation experiments, and fixed-work multi-accelerator scaling. Our aim is to make ideas about how models represent the world easier to turn into informative experiments, and those experiments easier to build upon.

2 Related Work

Representation-learning methods differ in the signals they learn from and how they connect information across modalities. DINOv2 learns visual features that capture semantic structure through self-supervision, without using labels or captions as training targets (Oquab et al., 2023). Predictive methods such as V-JEPA 2.1 learn spatial and temporal features from images and videos (Mur-Labadia et al., 2026). Paired observations supply relationships across modalities: early audiovisual learning and image-sentence alignment (Ngiam et al., 2011; Karpathy & Fei-Fei, 2015) precede the large-scale contrastive approaches of CLIP and ImageBind (Radford et al., 2021; Girdhar et al., 2023). Combining modalities within a model introduces further architectural choices. BLIP-2 and Idefics3 connect pretrained visual and language components (Li et al., 2023b; Laurençon et al., 2024), while Chameleon studies early fusion of image and text tokens (Chameleon Team, 2024). Qwen3-Omni extends multimodal modeling to text, image, audio, and video perception, with text and speech generation (Xu et al., 2025). These approaches locate cross-modal learning in different places: the supervision, the connections between components, and the representations learned jointly.

Language models trained on text have learned representations from which geographical coordinates and historical dates can be recovered with linear probes, generalizing to entities held out from probe training (Gurnee & Tegmark, 2024). Such evidence of learned world structure complements work connecting independently trained models. A learned linear projection from a frozen visual encoder into a frozen language model supports image captioning and visual question answering (Merullo et al., 2023). Increasing alignment between larger vision and language models motivates the Platonic Representation Hypothesis of a shared statistical model of reality (Huh et al., 2024). These findings make the route towards shared representations a concrete research problem: which learning signals, data relationships, and architectures encourage useful common structure? The relationship between alignment and performance depends on the shared and modality-specific information required by a task (Tjandrasuwita et al., 2025), motivating evaluation of both cross-modal transfer and retained capabilities.

Developing such methods involves choices about representational granularity, adaptation, and the cost of learning. ColBERT retains token-level embeddings for late-interaction retrieval (Khattab & Zaharia, 2020), while Matryoshka objectives train useful representations at multiple embedding dimensions (Kusupati et al., 2022). LoRA constrains parameter updates through trainable low-rank matrices (Hu et al., 2021), and gradient caching separates contrastive batch size from encoder activation memory (Gao et al., 2021). These choices can be composed in the training recipes evaluated here. Multimodal adaptation also raises the question of what to preserve. LiT keeps a pretrained image tower fixed while tuning a text tower to align with it (Zhai et al., 2022); jina-embeddings-v5-omni trains connectors between frozen encoders while preserving its text embeddings (Hönicke et al., 2026). Our adaptation study compares connector-only, LoRA, and full-parameter recipes through media retrieval and text-retrieval retention. Beyond retrieval, promptable vision-language representations use task context to support reinforcement learning (Chen et al., 2024), illustrating how the use of a representation can itself become a research choice. Learned representations can also supply objectives for search: Tian & Ha (2021) use CLIP similarity to guide evolutionary optimization of triangle-based drawings.

Research libraries make these choices accessible through reusable implementations and experimental workflows. Scenic supports vision and multimodal research with distributed JAX training (Dehghani et al., 2021), and solo-learn provides shared training and evaluation infrastructure for self-supervised visual learning (da Costa et al., 2022). LAVIS provides unified interfaces for language-vision models, datasets, training, and evaluation (Li et al., 2023a). Sentence Transformers supports embedding training and retrieval (Reimers & Gurevych, 2019), while PyLate extends this ecosystem with training and indexing for late-interaction models (Chaffin & Sourty, 2025). Representax provides a shared JAX foundation for developing, composing, and evaluating representation-learning methods across modalities. Its interfaces connect model and objective changes to shared data processing, training, and evaluation, allowing research methods to build on a common implementation.

3 REPRESENTAX

Research code should isolate the changes under investigation, reusing efficient, validated implementations for the rest of the system. Representax makes the model, learning procedure, data policy, and execution settings explicit, so researchers can inspect and revise the experiment they are running. Our vision for Representax is to make implementing a new idea require only the changes needed to express it, with minimal additional code to train, evaluate, and build upon the result.

3.1 CORE ABSTRACTIONS

Four abstractions are core to experiments in representation learning: models and processors, learning tasks, data distributions, and evaluators. Efficient shared infrastructure connects these components into complete experiments (Figure 1).

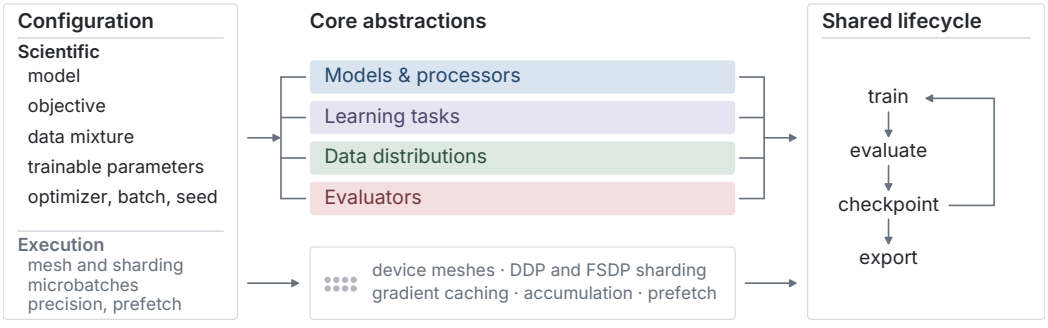


Figure 1: Representax at a glance. Scientific parameters define what is learned and build the four core abstractions; execution parameters set how the job runs. Both enter the shared lifecycle of training, evaluation, checkpointing, and export.

- **Models and processors.** We implement models as Equinox modules, which are JAX PyTrees with parameters and submodules stored in fields and forward computations defined

by Python methods (Kidger & Garcia, 2021). This keeps model structure and computation available for inspection and modification. Associated processors prepare inputs according to each model's supported modalities and processing requirements. The choice of how to update a model is similarly explicit: parameter selections follow its PyTree structure, allowing researchers to train selected components, introduce LoRA-based adaptation (Hu et al., 2021), or update the full model. Architectural changes that preserve the model and task interfaces can reuse the same training loop.

- **Learning tasks.** A learning task defines how examples and model outputs contribute to the training objective. Changing that objective changes the optimization problem, while the code that applies the selected optimizer, runs evaluations, and manages checkpoints can remain shared. Some tasks expose additional structure that supports composition: our Matryoshka implementation evaluates an objective at several embedding dimensions while reusing one encoder pass (Kusupati et al., 2022), and predictive tasks can specify updates to a target encoder. Keeping these behaviors with the task permits method-specific extensions while preserving useful code reuse.
- **Data distributions.** Data selection is an experimental choice, expressed through source definitions, example mappings, and replaceable sampling pipelines. Weighted mixtures are one built-in selection policy; caller-provided Grain datasets and iterators permit other policies to use the same loading and processing interfaces.
- **Evaluators.** Evaluators connect a learning method to established measures of representation quality. Fourteen implementations cover retrieval, similarity, classification, clustering, reward modeling, and predictive representation learning, among other tasks. Compatible evaluators can be composed over a shared batch stream, collecting multiple measurements in one traversal. Dataset adapters connect these implementations to BEIR-format retrieval data and pinned sources including Sprint Duplicate Questions, Banking77, Twenty Newsgroups, and CIFAR-100. Shared processing and metric implementations support comparisons under a consistent protocol, both during training and in standalone evaluation.

3.2 Source-Backed Data

Representax constructs training inputs from declared data sources without requiring a second, framework-specific dataset: source adapters provide access to records, example mappings interpret them for a learning task, and model-associated processors prepare the model inputs, so changing a data source mixture reuses the same processing and training code. Supported pipelines read records and decode media on demand, with bounded prefetch overlapping input preparation and accelerator computation. Built-in adapters cover local datasets and revision-pinned Hugging Face sources, referenced media can be fetched from local or HTTP(S) locations without staging them on the host, and custom readers and Grain iterators provide extension points for other sources and sampling policies. With fixed source contents and deterministic processing, supported loaders restore saved sampling and iterator state directly, without replaying consumed batches. Future research investigating learner-dependent data selection and curation can build upon this interface.

3.3 Configuring Experiments

A job configuration assembles the model, learning task, data distributions, and evaluators together with the optimizer, training schedule, and artifact settings. In Representax, configurations are typed Python objects with immutable fields and runtime validation through Pydantic. Hydra-Zen supports their composition and launch without requiring YAML. Experimental recipes can therefore live alongside their implementations as reviewable code, while each run records the resolved configuration used to construct it.

Within this configuration system, Representax distinguishes **scientific parameters**, which define the learning problem, from **execution parameters**, which describe how supported computations are carried out. Model and objective choices, data sampling, trainable parameters, and optimization schedules belong to the former; device placement, encoder microbatch sizes, and prefetch settings belong to the latter. The run record retains the complete configuration together with its scientific and execution components, making both the experimental recipe and its hardware-specific realization explicit.

The configured job enters a shared lifecycle covering training, scheduled evaluation, checkpointing, and inference export. Checkpoints preserve model and optimizer state, random state, training progress, and the data iterator position. Resume checks compare the recorded scientific configuration and data contract, distinguishing continuation of a run from changes to its method. Together with pinned code, data, and environments, these records provide the basis for reproducing an experiment and extending it through explicit changes.

3.4 Efficient Execution

Representax leverages JAX's flexible device meshes and array sharding to specify how computation and data are placed across accelerators. Training and evaluation operate on these model and batch arrays. For convenience, we provide DDP- and FSDP-style training configurations, while custom layouts can be expressed through explicit parameter partition rules and mesh definitions. These choices expose the tradeoffs between memory capacity and communication for a given workload and compute topology.

Execution strategies also exploit the structure of particular learning objectives. For supported contrastive tasks, Representax implements gradient caching to separate encoder activation memory from the configured contrastive batch (Gao et al., 2021). In our multiple-negatives ranking (MNR) implementation (Henderson et al., 2017), **encoder microbatches** are subsets processed during encoding and gradient replay; the loss still compares representations over the configured negative pool. This is an example of a task-specific execution path, with compatibility checks and tests against direct loss, gradient, and optimizer-update computations.

For objectives that decompose over examples or tokens, ordinary gradient accumulation provides a separate microbatching strategy. Both paths reuse the shared training lifecycle, while their applicability follows the structure of the task. This allows established execution techniques to support new experiments without treating every objective as computationally interchangeable.

4 Framework and Accelerator Comparisons

We evaluate our framework through paired training recipes against existing canonical implementations. In each paired throughput study, we run the same recipe in Representax and a reference framework. The comparisons span thirteen configurations across retrieval, sentence-pair learning, reward modeling, multimodal alignment, and predictive video learning, assessing execution performance across the methods our framework supports.

For these experiments, we used NVIDIA RTX 4090 GPUs, with one GPU per run, and a sixteen-chip TPU v5e allocation across four worker VMs. Within each hardware panel, we configure paired implementations with the same model, data, objective, training batch, optimization schedule, and precision policy, while allowing framework-specific execution choices. GPU references use PyTorch eager execution unless specified otherwise; TPU references use PyTorch/XLA (Paszke et al., 2019).

We run the same workload across five seeds for each configuration and measure warm training throughput, including input wait, gradient computation, synchronization, and parameter updates. Startup and checkpoint-related intervals are excluded; Appendix A gives the timing protocol and startup measurements.

The workloads cover the following training settings.

- **Dense retrieval.** MPNet on MS MARCO with multiple-negatives ranking loss and gradient caching, against Sentence Transformers (Reimers & Gurevych, 2019); an additional GPU control uses TorchInductor compilation.
- **Semantic similarity.** MPNet Base and BERT Base on STS Benchmark with cosine-similarity regression, against Sentence Transformers.
- **Pair classification.** The same encoders on duplicate-question pairs with a cosine contrastive objective, against Sentence Transformers.
- **Cross-encoder training.** MiniLM-L6 scoring query–passage pairs jointly, against the Sentence Transformers cross-encoder.

- **Late-interaction retrieval.** GTE-ModernColBERT with matched negative pools and mean-loss gradients, against PyLate (Chaffin & Sourty, 2025).
- **Outcome reward modeling.** Qwen3-0.6B on UltraFeedback with a Bradley–Terry objective and gradient accumulation, against a Torch/TRL reference (von Werra et al., 2020).
- **Process reward modeling.** The same model on step-level Math-Shepherd labels of at most 256 tokens, against a Torch/TRL reference.
- **Image–text alignment.** CLIP ViT-B/32 on COCO, against Sentence Transformers.
- **Audio–text alignment.** LoRA adaptation of LCO Omni 3B on AudioCaps, against a Sentence Transformers integration.
- **Video–text alignment.** The same adaptation on MSR-VTT, against a Sentence Transformers integration.
- **Predictive video learning.** ViT-B/16 V-JEPA 2.1 (Mur-Labadia et al., 2026) with a moving-average target encoder, against the official implementation; the GPU workload uses batch one and 24 videos.

Limitations. Some TPU contrastive recipes use device-local negative pools to match the reference integration, even though Representax supports global pools. These comparisons therefore evaluate a shared recipe rather than each framework's most capable configuration. Image–text and predictive-video timing also retain input-preparation and filesystem-cache differences, documented alongside the full execution settings in Appendix A.

4.1 Results

Table 1 reports absolute and relative throughput. Representax has the higher median in eleven of thirteen GPU configurations and in all thirteen TPU configurations, although TPU dense retrieval is effectively tied at 1.002x. Performance remains architecture-dependent: the BERT sentence-pair workloads favor the GPU reference, whereas their MPNet counterparts favor Representax.

Table 1: Warm throughput: median examples/s across five seeds; bold marks the higher median within a matched pair, not significance. R denotes Representax and ratios divide its median by the reference's. GPU columns compare one RTX 4090 against PyTorch eager (the dense TorchInductor control is in Appendix A.5); TPU columns compare a 16-chip v5e slice against PyTorch/XLA. Ratios compare frameworks within a hardware panel, not GPU against TPU. Lettered qualifications are defined in Appendix A.

	Single RTX 4090			16-chip v5e slice		
Workload	R ex/s	Ref. ex/s	R/Ref.	R ex/s	Ref. ex/s	R/Ref.
Dense retrieval	**305.35**	211.22	1.446	**2,510.21**	2,506.15	1.002
Similarity / MPNet	**559.72**	367.80	1.522	**4,045.86**	808.69	5.003
Similarity / BERT	286.21	**384.08**	0.745	**3,439.73**	829.87	4.145
Classification / MPNet	**552.81**	361.94	1.527	**4,124.54**	773.71	5.331
Classification / BERT	296.21	**382.92**	0.774	**3,478.86**	758.16	4.589
Cross-encoder	**670.37**	441.53	1.518	**2,701.60**	807.87	3.344
Late interaction [L]	**219.50**	55.18	3.978	**2,545.55**	1,593.27	1.598
Outcome reward [O]	**7.81**	5.43	1.439	**103.87**	66.29	1.567
Process reward [P,I]	**55.18**	16.79	3.287	**85.26**	9.94	8.578
Image-text [C]	**321.02**	106.22	3.022	**2,259.88**	1,062.06	2.128
Audio-text [M]	**2.66**	1.25	2.126	**11.24**	8.50	1.322
Video-text [M]	**3.44**	1.00	3.440	**88.66**	7.38	12.015
V-JEPA 2.1 [C,G]	**6.57**	1.30	5.040	**87.59**	59.63	1.469

The compiled dense control shows the weight of reference execution settings: at 0.964x the TorchInductor reference's throughput (Ansel et al., 2024), Representax outperforms eager Sentence Transformers and approaches GPU-specialized compiled execution using standard JAX/XLA, without GPU kernels written specifically for this recipe. Overall, these experiments demonstrate competitive

throughput across diverse training recipes, with substantial gains in many configurations. Appendix A presents absolute rates and paired loss trajectories, including residual differences in several TPU recipes; Appendix F examines padding, gradient replay, and compilation-cache reuse as contributors to execution cost.

5 End-to-End Representation Learning

We evaluate whether canonical representation-learning recipes produce useful held-out improvements when trained end to end in Representax: learning a transferable text retriever, improving image–text alignment, and adapting a multimodal model while retaining its existing capabilities. The studies comprise fifteen runs across three workload families, each recipe across three seeds, comparing the initial model with the final scheduled checkpoint without selecting checkpoints by their evaluation scores. These are finite-budget learning experiments, not tests of asymptotic convergence. A fourth family, late-interaction retrieval, did not improve under the recipe we applied; Appendix B.2 reports that retrieval regression and the follow-up it motivates.

5.1 Dense Retrieval and Transfer

This study asks whether contrastive retrieval fine-tuning in Representax turns a pretrained text encoder into a retriever, and whether the resulting representations transfer beyond the training collection. We train ETTIN Encoder 150M (Weller et al., 2025) on 488,576 duplicate-free MS MARCO query–passage pairs (Bajaj et al., 2016) using multiple-negatives ranking loss. Each pair is presented once, giving 3,817 updates at batch 128. Gradient caching supports the contrastive computation through the task interface described in Section 3.

We measure normalized discounted cumulative gain at rank ten (nDCG@10) on NanoMSMARCO, TREC DL 2019, and Natural Questions (Craswell et al., 2020; Kwiatkowski et al., 2019). NanoMSMARCO provides a small in-domain panel of 50 queries and 5,043 passages. The larger evaluations rank 43 TREC DL queries against the complete 8.84-million-passage MS MARCO corpus and 3,452 Natural Questions queries against 2.68 million documents. Natural Questions is used only for evaluation in this study, testing transfer beyond MS MARCO adaptation.

Mean nDCG@10 rises from 0.0730 to 0.5794 on NanoMSMARCO, from 0.0170 to 0.5404 on TREC DL, and from 0.0032 to 0.2922 on Natural Questions (Table 2). The gains therefore extend beyond the small NanoMSMARCO panel to full-corpus retrieval and a separate query distribution. They demonstrate useful adaptation, not state-of-the-art quality, and held-out evaluation does not exclude upstream pretraining contamination.

5.2 Image–Text Alignment

CLIP adaptation tests whether the same framework supports learning across modalities and transfer to a separate image–caption collection. We fine-tune CLIP ViT-B/32 (Radford et al., 2021) on 588,800 COCO image–caption pairs, covering 117,760 images with five distinct captions per image (Lin et al., 2014). A symmetric contrastive objective trains both retrieval directions over 1,150 updates at batch 512. Each caption is presented once, and no image is repeated within a training batch.

Evaluation uses a held-out Flickr30k panel with 1,000 images and 5,000 captions (Young et al., 2014), ranking the full panel in both directions. Mean nDCG@10 improves from 0.7373 to 0.8148 for text-to-image retrieval and from 0.7168 to 0.7665 for image-to-text retrieval (Table 2). Training on COCO thus improves alignment on Flickr30k rather than only reducing training loss; this is adaptation of a pretrained model, not a reproduction of CLIP pretraining. Appendix B gives the full settings for both studies.

Table 2: Held-out nDCG@10 before and after native adaptation. Final scores are means across three seeds, with the sample SD as a superscript; gain is final minus initial. The pretrained transfer baseline is shared across seeds; initial and final evaluations use the same complete corpora.

Model	Evaluation	Initial	Final	Gain
Dense	NanoMSMARCO	0.0730	$0.5794^{\pm 0.0111}$	+0.5064
CLIP	Flickr30k text → image	0.7373	$0.8148^{\pm 0.0023}$	+0.0775
CLIP	Flickr30k image → text	0.7168	$0.7665^{\pm 0.0025}$	+0.0497
Dense	TREC DL 2019	0.0170	$0.5404^{\pm 0.0082}$	+0.5234
Dense	Natural Questions	0.0032	$0.2922^{\pm 0.0011}$	+0.2889

5.3 Multimodal Adaptation and Retention

The multimodal study asks how adaptation can improve several modalities while preserving an existing text-retrieval capability. Using Jina v5 Omni Nano (Hönicke et al., 2026), we compose a symmetric contrastive task with Matryoshka objectives and compare three ways to update the model: connector-only adaptation trains the modality connectors and selected delimiter embeddings with the text pathway frozen; LoRA-based adaptation adds rank-eight LoRA adapters in the text, vision, and audio towers; full fine-tuning updates all parameters. Each recipe runs for 2,000 updates at batch 32, with three seeds per recipe. Training draws image–text, audio–text, and video–text pairs from COCO, AudioCaps, and MSR-VTT, respectively (Lin et al., 2014; Kim et al., 2019; Xu et al., 2016). The LoRA and full fine-tuning recipes also include MS MARCO text–text pairs. We evaluate text-to-media retrieval on Flickr30k, AudioCaps, and MSR-VTT, and text retention on NanoMSMARCO. Reporting these separately reveals whether media gains accompany a loss of text capability, rather than obscuring the tradeoff in a single aggregate score.

All three recipes improve retrieval in every media panel and every seed (Table 3). Connectors plus LoRA achieve the highest mean image and video scores, 0.7466 and 0.4984 nDCG@10; full fine-tuning achieves the highest audio score, 0.5028. Connector-only adaptation preserves the frozen text pathway's score of 0.6509. Text retrieval instead falls to 0.6121 with LoRA and 0.5994 with full fine-tuning, despite including text–text pairs in training. The study therefore demonstrates joint media improvements while making their relationship to text retention explicit.

Table 3: Change in nDCG@10 from each recipe's own initial checkpoint: mean over three seeds, sample SD as a superscript. Source mixtures and learning rates differ across recipes; connector-only training leaves the text pathway frozen. Absolute scores are in Appendix B.

Recipe	Flickr30k	AudioCaps	MSR-VTT	NanoMSMARCO
Connectors	$+0.1398^{\pm 0.0021}$	$+0.0313^{\pm 0.0011}$	$+0.1041^{\pm 0.0091}$	$+0.0000^{\pm 0.0000}$
Connectors + LoRA	$+0.1743^{\pm 0.0044}$	$+0.0427^{\pm 0.0010}$	$+0.1241^{\pm 0.0014}$	$-0.0387^{\pm 0.0043}$
Full fine-tuning	$+0.1264^{\pm 0.0063}$	$+0.0650^{\pm 0.0027}$	$+0.1002^{\pm 0.0034}$	$-0.0515^{\pm 0.0061}$

Limitations. These are comparisons of complete adaptation recipes, not a causal ablation of parameter selection: learning rates differ, and only the LoRA and full fine-tuning recipes include text–text training. Text retention uses the same 50-query NanoMSMARCO panel as the dense study, evaluation is text-anchored rather than any-to-any, and these tasks are not a complete multimodal benchmark suite. Within that scope, the results motivate methods that improve media retrieval without sacrificing text capabilities; Appendix B records absolute scores and the remaining settings.

6 Scaling and Sharding

Here we investigate how a fixed training job scales across multiple accelerators. Only execution parameters change with the device count: the mesh and the number of accumulation rounds per update. The training job itself, its global batch, tokens, and masks, is identical at every device count. The study therefore exercises, on hardware, the separation between scientific and execution parameters described in Section 3.

We train a controlled 505-million-parameter ModernBERT masked-language model (Warner et al., 2024; Devlin et al., 2019) on 1, 2, 4, and 8 A100 SXM4 40-GB GPUs on a single node with replicated data parallelism. Each update processes 256 sequences of 512 tokens (131,072 tokens); a local microbatch of 16 with 16, 8, 4, and 2 accumulation rounds preserves that global batch, and token and masking hashes match within each seed across device counts. Three seeds each run 21 updates, excluding the first from warm rates. The model is randomly initialized and trained on WikiText-2 text (Merity et al., 2016) repeated to fill the run: a pretraining-like workload for measuring execution, not a pretraining run, so we neither train to convergence nor evaluate the resulting model.

Throughput rises from 32,015 to 219,975 tokens/s (Figure 2). Mean within-seed speedups are 1.968x, 3.810x, and 6.871x on 2, 4, and 8 GPUs, corresponding to 98.4%, 95.2%, and 85.9% parallel efficiency. All updates are finite and non-skipped, and the compiled executable footprint stays between 32.5 and 33.6 GiB per device. BF16 loss trajectories are close but not identical across device counts, with a maximum cross-device loss spread of 0.0523.

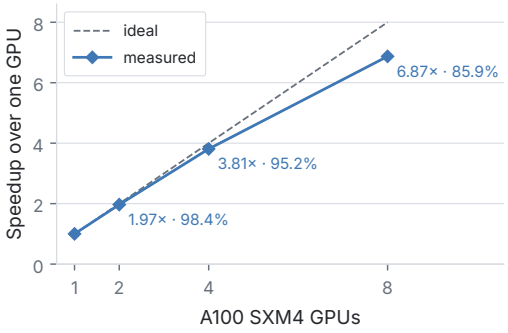


Figure 2: Fixed-work strong scaling on one A100 node. Faint lines are the three seeds; diamonds are means. Speedup is relative to each seed's one-GPU rate, and each annotation gives speedup and parallel efficiency. Warm timings exclude compilation and cache loading.

Limitations. This is strong scaling of replicated execution on one node, measured over 20 warm updates per run. FSDP capacity gains remain unmeasured, and multi-node GPU execution remains untested. Absolute rates, memory, and compilation costs appear in Appendix C.

7 Conclusion

Representax provides a shared foundation for representation-learning research, connecting flexible method development with the infrastructure needed to train, evaluate, and scale experiments. Its separation of scientific choices from execution strategies allows researchers to investigate learning methods while adapting supported workloads to available hardware. This separation also opens a path to hardware-aware autotuning, selecting execution configurations for a fixed training recipe and compute environment while preserving the intended learning computation. Our evaluations demonstrate competitive training throughput across a range of GPU and TPU workloads, improvements in held-out retrieval through adaptation, and efficient fixed-work scaling to eight accelerators.

These results provide a starting point for research towards natively multimodal systems. Our adaptation studies motivate a concrete challenge: jointly improving representations across modalities while preserving or strengthening existing text capabilities, rather than relying on a frozen text pathway. Beyond fixed training recipes, future work could investigate online representation learning, reinforcement learning that shapes representations through interaction, evolutionary search over architectures and learning objectives, and adaptive datasets whose composition, sampling, or generation changes with the learner. Indeed, some of these approaches may draw inspiration from biological systems, informing how models integrate sensory information, learn from experience, and adapt to changing environments. We intend Representax to make these ideas practical to develop, compare, and refine. With the community, we aim to build on this foundation to investigate a shared question: how should models represent and understand the world?

Reproducibility Statement

The repository contains numbered experiment launchers, configuration objects, per-run source revisions, data and checkpoint revisions, and content hashes. The paper's frozen evidence includes the measured update records, and a historical-methods inventory records the settings actually used; every numerical table and plot derives from these records. Rerunning training requires the recorded accelerator configurations, environments, and upstream datasets and checkpoints. Appendix E lists these artifacts and their scope.

Ethics Statement

The experiments reuse public datasets and checkpoints under their respective terms; the software's license does not supersede those terms. Representation models may preserve biases or sensitive information in their training data. The reported retrieval metrics do not establish fairness, privacy, or suitability for high-stakes deployment.

AI Use Statement

Generative AI tools assisted implementation, debugging, experiment design, analysis, literature discovery, and manuscript drafting. The human author is responsible for the final contents and claims.

References

Jason Ansel, Edward Yang, Horace He, et al. PyTorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024. URL https://docs.pytorch.org/assets/pytorch2-2.pdf.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html.

Payal Bajaj et al. MS MARCO: A human generated MAchine Reading COmprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016. URL https://arxiv.org/abs/1611.09268.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. URL https://arxiv.org/abs/1206.5538.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. Software, 2018. URL https://github.com/jax-ml/jax.

Antoine Chaffin and Raphaël Sourty. PyLate: Flexible training and retrieval for late interaction models. *arXiv preprint arXiv:2508.03555*, 2025. URL https://arxiv.org/abs/2508.03555.

Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024. URL https://arxiv.org/abs/2405.09818.

William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-language models provide promptable representations for reinforcement learning. *arXiv preprint arXiv:2402.02651*, 2024. URL https://arxiv.org/abs/2402.02651.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. Overview of the TREC 2019 deep learning track. *arXiv preprint arXiv:2003.07820*, 2020. URL https://arxiv.org/abs/2003.07820.

Victor Guilherme Turrisi da Costa, Enrico Fini, Moin Nabi, Nicu Sebe, and Elisa Ricci. solo-learn: A library of self-supervised methods for visual representation learning. *Journal of Machine Learning Research*, 23(56):1–6, 2022. URL https://www.jmlr.org/papers/v23/21-1155.html.

DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX ecosystem. Software, 2020. URL https://github.com/google-deepmind. Project-recommended citation for Optax, https://github.com/google-deepmind/optax.

Mostafa Dehghani, Alexey Gritsenko, Anurag Arnab, Matthias Minderer, and Yi Tay. Scenic: A JAX library for computer vision research and beyond. *arXiv preprint arXiv:2110.11403*, 2021. URL https://arxiv.org/abs/2110.11403.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019. URL https://arxiv.org/abs/1810.04805.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980. doi: 10.1007/BF00344251. URL https://link.springer.com/article/10.1007/BF00344251.

Colin Gaffney, Shutong Li, Daniel Ng, Anastasia Petrushkina, Niket Kumar, Adam Cogdell, Mridul Sahu, Yaning Liang, Nikhil Bansal, Justin Pan, Angel Mau, Abhishek Agrawal, Marco Berlot, Ruoxin Sang, Kiranbir Sodhia, and Rakesh Iyer. Orbax: Distributed checkpointing with JAX. *arXiv preprint arXiv:2605.23066*, 2026. URL https://arxiv.org/abs/2605.23066.

Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. Scaling deep contrastive learning batch size under memory limited setup. *arXiv preprint arXiv:2101.06983*, 2021. URL https://arxiv.org/abs/2101.06983.

Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. ImageBind: One embedding space to bind them all. In *Proceedings of CVPR*, 2023. URL https://arxiv.org/abs/2305.05665.

Wes Gurnee and Max Tegmark. Language models represent space and time. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/0a6059857ae5c82ea9726ee9282a7145-Abstract-Conference.html.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. URL https://arxiv.org/abs/1803.10122.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*, 2017. URL https://arxiv.org/abs/1705.00652.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL https://www.cs.toronto.edu/~rsalakhu/papers/science.pdf.

Florian Hönicke, Michael Günther, Andreas Koukounas, Mohammad Kalim Akram, Saba Sturua, and Han Xiao. jina-embeddings-v5-omni: Geometry-preserving embeddings via locked aligned towers. *arXiv preprint arXiv:2605.08384*, 2026. URL https://arxiv.org/abs/2605.08384.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. Position: The Platonic Representation Hypothesis. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20617–20642. PMLR, 2024. URL https://proceedings.mlr.press/v235/huh24a.html.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of CVPR*, 2015. URL https://arxiv.org/abs/1412.2306.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of SIGIR*, 2020. URL https://arxiv.org/abs/2004.12832.

Patrick Kidger and Cristian Garcia. Equinox: Neural networks in JAX via callable PyTrees and filtered transformations. *arXiv preprint arXiv:2111.00254*, 2021. URL https://arxiv.org/abs/2111.00254.

Chris Dongjoo Kim, Byeongchang Kim, Hyunmin Lee, and Gunhee Kim. AudioCaps: Generating captions for audios in the wild. In *Proceedings of NAACL-HLT*, pp. 119–132, 2019. URL https://aclanthology.org/N19-1011/.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. *arXiv preprint arXiv:2205.13147*, 2022. URL https://arxiv.org/abs/2205.13147.

Tom Kwiatkowski et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. URL https://aclanthology.org/Q19-1026/.

Hugo Laurençon, Andrés Marafioti, Victor Sanh, and Léo Tronchon. Building and better understanding vision-language models: Insights and future directions. *arXiv preprint arXiv:2408.12637*, 2024. URL https://arxiv.org/abs/2408.12637.

Dongxu Li, Junnan Li, Hung Le, Guangsen Wang, Silvio Savarese, and Steven C.H. Hoi. LAVIS: A one-stop library for language-vision intelligence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 31–41, 2023a. doi: 10.18653/v1/2023.acl-demo.3. URL https://aclanthology.org/2023.acl-demo.3/.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19730–19742, 2023b. URL https://proceedings.mlr.press/v202/li23q.html.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of ECCV*, 2014. URL https://arxiv.org/abs/1405.0312.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. URL https://arxiv.org/abs/1609.07843.

Jack Merullo, Louis Castricato, Carsten Eickhoff, and Ellie Pavlick. Linearly mapping from image to text space. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2209.15162.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. URL https://arxiv.org/abs/1301.3781.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. URL https://arxiv.org/abs/2210.07316.

Lorenzo Mur-Labadia, Matthew Muckley, Amir Bar, Mido Assran, Koustuv Sinha, Mike Rabbat, Yann LeCun, Nicolas Ballas, and Adrien Bardes. V-JEPA 2.1: Unlocking dense features in video self-supervised learning. *arXiv preprint arXiv:2603.14482*, 2026. URL https://arxiv.org/abs/2603.14482.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, 2011. URL https://ai.stanford.edu/~jngiam/papers/NgiamKhoslaKimNamLeeNg2011.pdf.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, et al. DINOv2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. URL https://arxiv.org/abs/2304.07193.

Adam Paszke et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1912.01703.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of ICML*, 2021. URL https://arxiv.org/abs/2103.00020.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of EMNLP-IJCNLP*, 2019. URL https://arxiv.org/abs/1908.10084.

Marvin Ritter, Ihor Indyk, Aayush Singh, Andrew Audibert, Anoosha Seelam, Camelia Hanes, Eric Lau, Jacek Olesiak, Jiyang Kang, and Xihui Wu. Grain: Feeding JAX models. Software, 2023. URL https://github.com/google/grain.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of CVPR*, 2022. URL https://arxiv.org/abs/2112.10752.

Richard S. Sutton. The bitter lesson, March 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Yingtao Tian and David Ha. Modern evolution strategies for creativity: Fitting concrete images and abstract concepts. *arXiv preprint arXiv:2109.08857*, 2021. URL https://arxiv.org/abs/2109.08857.

Megan Tjandrasuwita, Chanakya Ekbote, Liu Ziyin, and Paul Pu Liang. Understanding the emergence of multimodal representation alignment. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 59723–59760. PMLR, 2025. URL https://proceedings.mlr.press/v267/tjandrasuwita25a.html.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers reinforcement learning. Open-source software, 2020. URL https://github.com/huggingface/trl.

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. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *arXiv preprint arXiv:2412.13663*, 2024. URL https://arxiv.org/abs/2412.13663.

Orion Weller, Kathryn Ricci, Marc Marone, Antoine Chaffin, Dawn Lawrie, and Benjamin Van Durme. Seq vs seq: An open suite of paired encoders and decoders. *arXiv preprint arXiv:2507.11412*, 2025. URL https://arxiv.org/abs/2507.11412.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

Jin Xu, Zhifang Guo, Hangrui Hu, Yunfei Chu, et al. Qwen3-Omni technical report. *arXiv preprint arXiv:2509.17765*, 2025. URL https://arxiv.org/abs/2509.17765.

Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A large video description dataset for bridging video and language. In *Proceedings of CVPR*, 2016. URL https://openaccess.thecvf.com/content_cvpr_2016/html/Xu_MSR-VTT_A_Large_CVPR_2016_paper.html.

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014. URL https://aclanthology.org/Q14-1006/.

Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. LiT: Zero-shot transfer with locked-image text tuning. In *Proceedings of CVPR*, 2022. URL https://arxiv.org/abs/2111.07991.

Table 4: Timing-suite configuration. Batch is global GPU/TPU batch. Lengths are padded caps; q/d denotes query/document. Full revisions and per-run settings are in the historical-methods inventory.

Recipe	Model	Input shape	Batch
Dense retrieval	MPNet Base	q32/d256	2048/2048
Similarity (two rows)	MPNet Base / BERT Base	128 tokens	256/256
Classification (two rows)	MPNet Base / BERT Base	128 tokens	256/256
Cross-encoder	MiniLM-L6	Up to 512 tokens	128/128
Late interaction [L]	GTE-ModernColBERT	q32/d256	512/512
Outcome reward	Qwen3-0.6B	1,024 tokens/sequence	128/128
Process reward [P]	Qwen3-0.6B	GPU 256 / TPU 2,048; ≤ 256 valid	64/64
Image-text [C]	CLIP ViT-B/32	224^2 pixels	512/512
Audio-text	LCO Omni 3B	10 s, 16 kHz	32/48
Video-text	LCO Omni 3B	16 frames, 224^2 pixels	2/112
V-JEPA 2.1 [C,G]	ViT-B/16 + predictor	16 frames, 256^2 pixels	1/128

A Paired Workloads and Timing

A.1 Models, Shapes, and Data

The paired panel uses seeds 7, 42, 773, 1234, and 2026: 130 runs per backend, plus five compiled GPU dense-retrieval controls. Table 4 describes the actual timing workloads. These bounded datasets often cycle during 22 updates; their size is not the number of unique examples consumed by the training job. A fixed padded shape is also not a count of valid tokens.

Text tasks. Dense retrieval uses the pinned Sentence Transformers MPNet checkpoint and a 65,536-pair duplicate-free MS MARCO table, with seed-specific permutations. Its native encoder chunks are 32; the GPU reference uses 128. The TPU reference uses local batches of 128 and local negative pools. Similarity uses STS Benchmark pairs and cosine regression; classification uses a bounded duplicate-question collection and a cosine contrastive objective with margin 0.5. Cross-encoder training uses binary labels from 3,840 pairs involving 1,920 queries. Historical late interaction uses 15,360 source-order pairs with repeated query-positive examples, temperature 0.02, and native chunk eight. It must not be conflated with the revised hard-negative learning experiment.

Reward tasks. Outcome reward uses Bradley–Terry preference training on 3,840 UltraFeedback rows. Process reward uses binary process labels from Math-Shepherd, with examples selected for four steps and mixed labels. The latter's preparation bound is 256 tokens. Both frameworks execute padded length 256 on GPU and 2,048 on TPU. The original GPU reference incorrectly used 2,048 while native execution used 256; the corrected reference rerun retains the same data, checkpoint, five seeds, optimizer, and batch settings. Its actual training tensors are checked as microbatch two by length 256. The replacement process-reward panel also shares the initial one-row scalar head. GPU timing uses identical update indices in both frameworks, excluding 1, 2, and the post-checkpoint update 12; TPU timing excludes 1 and 2. Original measurements remain in the artifact correction history, not the active throughput table. GPU native accumulation uses microbatches four and two, respectively, to reach the stated global batches. Both use peak learning rate 10^{-5}. Historical outcome reward loaded a BF16 base in both frameworks, but only native initialization promoted trainable leaves to FP32 before creating optimizer state. The replacement GPU control promotes the same BF16-rounded base and shared scalar head to FP32 in both frameworks; forward computation remains BF16. Each framework uses microbatch four and 32 accumulation rounds for global batch 128. The independently initialized historical heads and BF16 reference optimizer are not the active GPU comparison. The corrected TPU control uses the same precision and shared-head policy, with global batch 128. Its reference uses microbatch two and four local accumulation rounds; native execution retains its recorded configuration. The TPU reference averages parameter gradients before clipping and checks full trainable-parameter identity across ranks outside measured timing. Historical process-reward reference construction preceded Trainer seeding and selected one score from a newly initialized classification head without an explicit one-label configuration. The replacement controls correct both issues.

Multimodal tasks. Image-text uses 3,840 COCO images with four repeated presentations of the first caption per image in the historical timing recipe. The longer CLIP study instead uses distinct captions. Audio-text uses 1,920 AudioCaps clips, repeated four times; video-text uses 3,840 MSR-VTT videos. Both adapt rank-four, alpha-eight text-tower LoRA in the 3B LCO model, with the base in BF16. Both corrected GPU panels use encoder chunks of one. Historical native TPU runs used chunks of two; the corrected TPU settings are detailed below. TPU reference execution uses direct MNR. These are heavier multimodal workloads than text-only encoding, not an equal-compute comparison across modalities. Native trainable adapters are FP32, whereas the historical Transformers (Wolf et al., 2020) `add_adapter` path leaves them BF16. Native A matrices use a normal distribution with standard deviation $d_{in}^{-1/2}$; PEFT defaults to a uniform distribution on $[-d_{in}^{-1/2}, d_{in}^{-1/2}]$. Both initialize B to zero, but this does not align subsequent optimization. The historical reference also constructs these adapters before Trainer applies its seed.

For historical TPU audio specifically, batch 48 gives three examples per chip. Native and reference runs both use device-local negative pools: each audio query is compared against three captions, not all 48 captions in the optimizer batch. We matched the reference runner's configured objective; its existing PJRT integration did not connect Sentence Transformers' cross-device embedding gather. This is a limitation of that integration, not of the Sentence Transformers MNR objective or its GPU distributed support. Gradient synchronization alone does not expand the contrastive candidate pool.

Native rematerialized GradCache uses chunks of two, padding each device's encoder input to four slots and discarding the extra representation before the loss. The reference directly encodes its three examples. Both enable layer activation checkpointing; native execution additionally rematerializes the encoder chunks. These are execution differences, not a change in the number of examples or negatives contributing to the objective. All five historical native runs are slower. Padding and replay are plausible contributors, but no controlled direct-versus-cached TPU audio ablation isolates their effect on throughput.

The corrected audio panel uses direct MNR in both frameworks with all 48 captions as candidates: each reference rank scores its three queries against the complete gathered pool. Both use FP32 adapters and optimizer moments, the same initialization distribution, and zero B matrices. Native and reference median rates are 11.24 and 8.50 examples/s. This replacement changes the negative pool and execution together; it does not isolate the historical padding overhead. All five reference runs pass final sixteen-rank adapter identity checks.

The corrected TPU video panel retains global batch 112 and device-local pools of seven in both frameworks. Native GradCache uses chunk seven without padding; the reference directly encodes seven examples per rank. Both retain layer activation checkpointing and use FP32 adapters and optimizer moments. All five reference runs pass final sixteen-rank parameter identity. Reference seeds 773, 1234, and 2026 ran on a replacement allocation in `us-west1-c`; the other corrected video cells ran in `us-central1-a`. Both allocations use sixteen v5e chips in a 4x4 topology and the same locked training environments. This is a same-configuration comparison across documented allocations, not a claim that all cells occupied the same physical slice.

Predictive video. V-JEPA uses a 12-layer, width-768 ViT-B/16 encoder, 12-layer width-384 predictor, tubelet size two, and supervision at layers 2, 5, 8, and 11. The EMA coefficient is 0.99925. The initialization and short schedule define an execution test, not a released checkpoint's quality evaluation. GPU and TPU use 24 and 2,816 Something-Something-v2 training videos, respectively. Per-run manifests establish this difference; a later shared asset directory is not authoritative for an earlier run.

Except for the noted reward and predictive-video schedules, paired tasks use AdamW with peak learning rate 2×10^{-5}, zero weight decay, and gradient clipping at one; late interaction uses 3×10^{-6}. Most use one warmup update followed by cosine decay across 22 updates. V-JEPA instead uses weight decay 0.04, no clipping, and a linear 10^{-4} to 6×10^{-4} schedule over 12,000 updates. Native matrix computation is BF16 with FP32 accumulation in the recorded precision policy. Frozen multimodal bases and outcome-reward loading have the exceptions described above.

A.2 Reference Execution and Environments

Sentence Transformers supplies dense, sentence-pair, cross-encoder, and contrastive multimodal reference paths. PyLate supplies the late-interaction path. Reward adapters use the Torch/TRL ecosystem (von Werra et al., 2020); predictive video uses the official V-JEPA implementation. Launch, data, and metric adapters make these programs fit the shared measurement contract. Thus *reference* means the documented upstream-based integration, not that every upstream training loop was used unchanged.

GPU comparisons use PyTorch eager unless explicitly marked TorchInductor. TPU references use PyTorch/XLA 2.9.0 and PJRT. The same program launches on all four worker VMs, yielding sixteen ranks. Native JAX uses four distributed processes, each controlling four local TPU devices. This is observed multi-host execution, not multi-node GPU scaling. Sentence Transformers' cross-device-negative helper was not connected to an initialized Torch distributed group in the historical integration; its paired contrastive recipes generally used per-device negatives, with the late-interaction mismatch identified above. The corrected late-interaction reference instead uses a global pool through layout-pinned XLA collectives. Its gather passes real-tensor identity and normalization checks on all sixteen ranks; its backward reduction includes remote-query contributions and passes a same-backend global-batch gradient/update oracle. The earlier unpinned reference fails the real-tensor check and is not used for the corrected table. Corrected audio initializes the Torch distributed group and uses Sentence Transformers' global embedding gather; its observed score matrix is three by 48 on each rank. The corrected TPU reference integration explicitly averages functional gradient outputs before clipping and updating, checks replica identity, and requests BF16 autocast without downcasting FP32 master weights or optimizer state. Its post-backward execution boundary is included in measured step time. These are implementation-level comparisons, not isolated compiler benchmarks.

The native environment records JAX/jaxlib 0.11.1, Equinox 0.13.8, Grain 0.2.18, Optax 0.2.8, and Orbax 0.12.4. GPU references record PyTorch 2.11.0+cu128; TPU references record PyTorch 2.9.0+cpu and Torch/XLA 2.9.0. Sentence Transformers is 5.6.1 in the principal environment; the PyLate environment uses 5.3.0, alongside PyLate 1.6.0. Transformers versions differ between these environments. Rather than assigning one invented common version, the supplementary inventory preserves each run's complete environment.

A.3 Pairing Audit and Short-Run Checks

The original late-interaction comparison [L] used duplicated pairs with sequential native batches but shuffled GPU reference batches. Its cached reference backpropagated summed rather than mean gradients. Native TPU execution also ignored the configured local-negative scope, comparing 512 candidates against the reference's 32. The corrected GPU panel uses shared duplicate-free data, sequential batches, 32/256-token limits, and mean-loss cached gradients. Initial NanoMSMARCO nDCG@10 is 0.7123 versus 0.7089; mean final scores are 0.7151 versus 0.7089. GPU reference trajectories are identical across seeds under this deterministic schedule. Both corrected TPU frameworks use global pools of 512; final losses are 0.02067 and 0.02254, with deterministic histories within each framework.

Outcome reward [O] aligns FP32 master weights and moments as described above. Final paired losses differ by at most 0.0081 on GPU and 0.0057 on TPU. Process reward [I] aligns scalar-head initialization and applies seeds before model construction. Every corrected TPU reference verifies identical initial and final trainable parameters across all sixteen ranks. These checks support paired execution, not equivalence of long-run learning.

Corrected audio/video adapters [M] use FP32 trainable parameters and moments, seeded normal A matrices at the same scale, and zero B matrices. The random A draws are independent across frameworks, not bitwise shared. Final paired TPU losses differ by at most 0.0544 for audio and 0.0055 for video. Audio's replacement changes both the negative pool and execution method; video's native chunk seven removes padding while preserving seven local candidates. Neither comparison isolates a single source of performance improvement.

Two input-cache qualifications [C] remain: TPU image-text and V-JEPA show first-seed cold-filesystem effects, and reference pre-materialization moves some preprocessing outside the training

Table 5: Absolute warm training rates on one RTX 4090. Columns show median per-seed examples/s (ex/s) and optimizer steps/s (st/s); ratio is native/reference examples/s. Batch is global examples per update. Ratios are shown only for validated matched protocols; historical discrepancies and replacements are documented in the text. Qualifications are defined in the text.

Recipe	Batch	Native ex/s	st/s	Ref. ex/s	st/s	Ratio
Dense retrieval	2048	305.35	0.149	211.22	0.103	1.446
Similarity / MPNet	256	559.72	2.186	367.80	1.437	1.522
Similarity / BERT	256	286.21	1.118	384.08	1.500	0.745
Classification / MPNet	256	552.81	2.159	361.94	1.414	1.527
Classification / BERT	256	296.21	1.157	382.92	1.496	0.774
Cross-encoder	128	670.37	5.237	441.53	3.449	1.518
Late interaction [L]	512	219.50	0.429	55.18	0.108	3.978
Outcome reward [O]	128	7.81	0.061	5.43	0.042	1.439
Process reward [P,I]	64	55.18	0.862	16.79	0.262	3.287
Image-text [C]	512	321.02	0.627	106.22	0.207	3.022
Audio-text [M]	32	2.66	0.083	1.25	0.039	2.126
Video-text [M]	2	3.44	1.719	1.00	0.500	3.440
V-JEPA 2.1 [C,G]	1	6.57	6.571	1.30	1.304	5.040
Dense / Inductor	2048	305.35	0.149	316.61	0.155	0.964

Table 6: Absolute warm training rates on the full 16-chip v5e slice. Columns show median per-seed examples/s (ex/s) and optimizer steps/s (st/s); ratio is native/reference examples/s. Batch is global examples per update. Ratios are shown only for validated matched protocols; historical discrepancies and replacements are documented in the text. Qualifications are defined in the text.

Recipe	Batch	Native ex/s	st/s	Ref. ex/s	st/s	Ratio
Dense retrieval	2048	2,510.21	1.226	2,506.15	1.224	1.002
Similarity / MPNet	256	4,045.86	15.804	808.69	3.159	5.003
Similarity / BERT	256	3,439.73	13.436	829.87	3.242	4.145
Classification / MPNet	256	4,124.54	16.111	773.71	3.022	5.331
Classification / BERT	256	3,478.86	13.589	758.16	2.962	4.589
Cross-encoder	128	2,701.60	21.106	807.87	6.311	3.344
Late interaction [L]	512	2,545.55	4.972	1,593.27	3.112	1.598
Outcome reward [O]	128	103.87	0.811	66.29	0.518	1.567
Process reward [P,I]	64	85.26	1.332	9.94	0.155	8.578
Image-text [C]	512	2,259.88	4.414	1,062.06	2.074	2.128
Audio-text [M]	48	11.24	0.234	8.50	0.177	1.322
Video-text [M]	112	88.66	0.792	7.38	0.066	12.015
V-JEPA 2.1 [C,G]	128	87.59	0.684	59.63	0.466	1.469

timer. The V-JEPA GPU/TPU batch and data differences [G] are recorded above. Process reward's short valid inputs [P], even when padded to 2,048 on TPU, do not represent long-reasoning training.

A.4 Absolute Rates and Startup

The other historical TPU contrastive pairs use per-device negatives in both frameworks, while a single-GPU negative pool covers the full batch. GPU replacement jobs each use one device, with up to four concurrent jobs sharing host resources; per-seed timing variation is retained.

Optimizer steps/s uses the same measured-time denominator as examples/s. Consecutive updates are not independent replicates. Historical native GPU late interaction retains 15 usable intervals after first-use exclusions. Corrected GPU pairs retain the same 19 update indices after excluding first use and the post-checkpoint interval, whose reference timer includes save/resume overhead. The evidence archive preserves the replaced measurements.

Tables 5 and 6 report absolute rates. Because a batch can mean retrieval pairs, preference pairs, or videos, examples/s should be compared within a recipe, not averaged across recipes. Different GPU/TPU batch sizes and allocations preclude an equal-cost or equal-work hardware ranking.

Table 7: Recorded early-step diagnostics on one RTX 4090, median seconds over five seeds. Native first-use is the sum of dispatch-to-completion intervals across the reported number of events, including new shapes and resumed execution. Reference columns are complete first and second optimizer-step intervals, including input wait. They are different timing boundaries, not a compilation-speed comparison; neither isolates pure compilation or full startup. Cache states vary. A dash means unrecorded, not zero, or no separate native Inductor run.

Recipe	Native first-use (s)	Events	Ref. step 1 (s)	Ref. step 2 (s)
Dense retrieval	26.2	1	11.2	9.7
Similarity / MPNet	17.3	1	1.5	0.6
Similarity / BERT	5.9	1	1.5	0.6
Classification / MPNet	17.6	1	1.5	0.7
Classification / BERT	6.0	1	1.4	0.6
Cross-encoder	6.3	2	0.8	0.2
Late interaction [L]	37.5	2	10.0	9.3
Outcome reward [O]	37.9	2	23.9	23.5
Process reward [P,I]	5.6	2	4.7	4.0
Image-text [C]	9.8	2	15.3	4.8
Audio-text [M]	18.8	1	48.9	20.8
Video-text [M]	9.1	2	2.5	1.5
V-JEPA 2.1 [C,G]	6.2	2	1.2	0.8
Dense / Inductor	—	—	45.9	6.5

Table 8: Recorded early-step diagnostics on the full 16-chip v5e slice, median seconds over five seeds. Native first-use is the sum of dispatch-to-completion intervals across the reported number of events, including new shapes and resumed execution. Reference columns are complete first and second optimizer-step intervals, including input wait. They are different timing boundaries, not a compilation-speed comparison; neither isolates pure compilation or full startup. Cache states vary. A dash means unrecorded, not zero, or no separate native Inductor run.

Recipe	Native first-use (s)	Events	Ref. step 1 (s)	Ref. step 2 (s)
Dense retrieval	288.2	1	42.5	45.1
Similarity / MPNet	185.4	1	29.1	31.8
Similarity / BERT	11.1	1	29.9	33.2
Classification / MPNet	185.6	1	29.2	31.8
Classification / BERT	11.1	1	30.0	33.3
Cross-encoder	9.3	1	24.5	26.2
Late interaction [L]	52.4	1	52.2	4.6
Outcome reward [O]	7.6	1	175.1	13.4
Process reward [P,I]	7.1	1	96.4	37.5
Image-text [C]	9.8	1	42.0	44.3
Audio-text [M]	11.7	1	271.0	44.2
Video-text [M]	26.1	1	1,057.1	56.7
V-JEPA 2.1 [C,G]	35.3	1	151.0	150.5

Tables 7 and 8 distinguish native first-use intervals from the reference's first two optimizer-step intervals. Native first-use records time dispatch through completion for a new input signature, including compilation or cache loading and an initial update; their sums can include later shape changes and checkpoint-resume events. Reference step intervals include input wait and do not isolate compilation. All 135 reference runs retain both early intervals; all 130 native runs retain first-use events and separate setup timings. These measurements are not interchangeable startup costs or pure compiler benchmarks. The paired suite was not a balanced randomized cold-cache experiment. Per-seed early intervals, setup records, warm step-time coefficients of variation, and rates are preserved with the frozen evidence.

A.5 Compiled GPU Reference

TorchInductor is the compilation backend used by PyTorch's `torch.compile`; it is not a separate reference framework. The matched suite includes five compiled Sentence Transformers runs

for MPNet dense retrieval at global batch 2,048 on one RTX 4090. Median throughput is 316.61 examples/s, against Representax's 305.35 and eager Sentence Transformers' 211.22. Thus the native rate is 0.964x the compiled reference, despite its 1.446x eager-relative rate. Table 5 retains this control as the Dense / Inductor row. No matched Inductor measurements are available for the other recipes. The main table's bold values therefore identify winners against its stated references, not against every possible compiled configuration.

A.6 Paired Training Trajectories

Figure 3 compares GPU training losses over all 22 optimizer updates, including updates excluded from throughput timing. The five configured seeds are 7, 42, 773, 1234, and 2026; deterministic recipes can produce overlapping curves. We inspect the shape and scale of the trajectories rather than require pointwise equality or infer long-run convergence from these short runs.

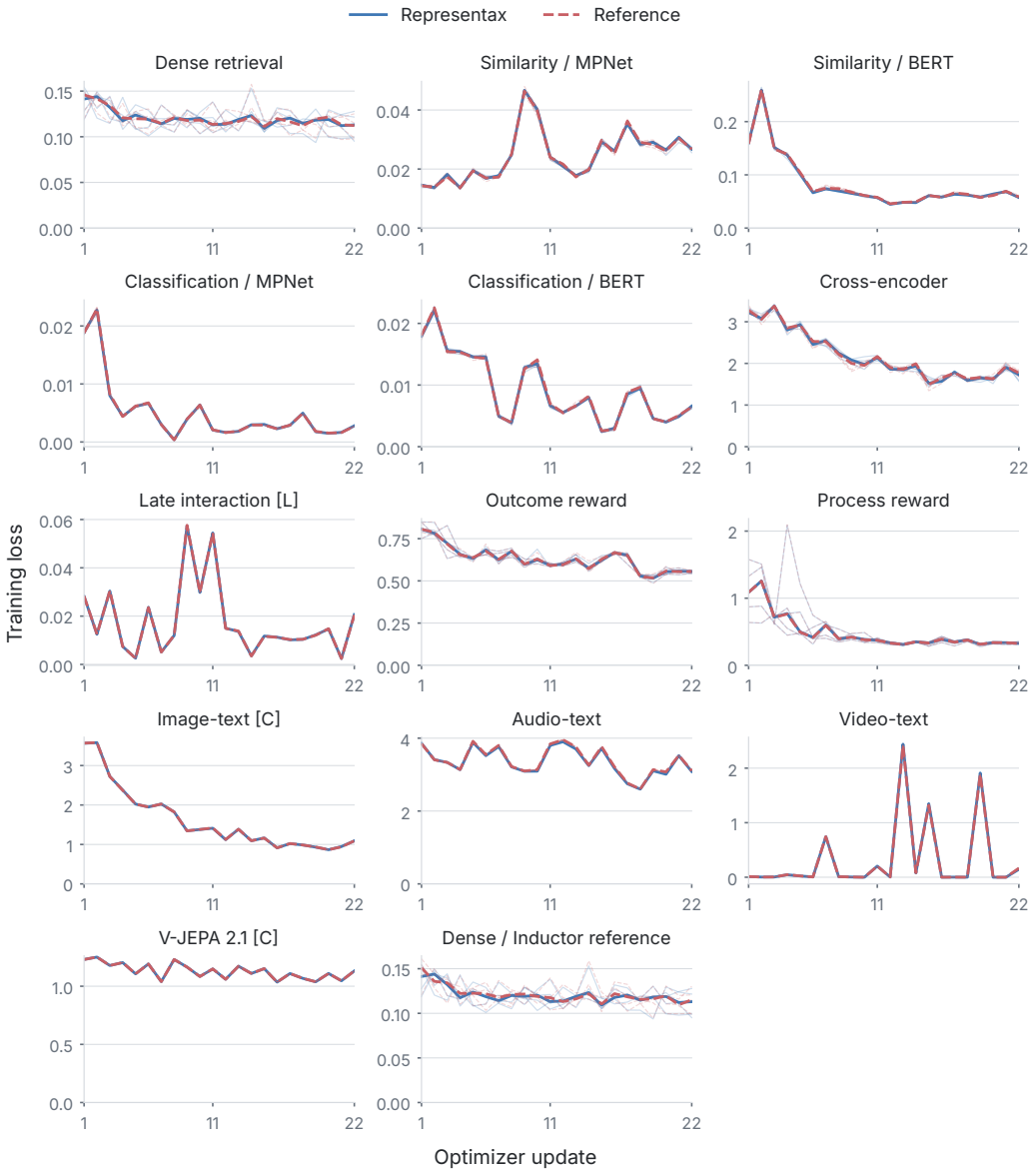


Figure 3: Paired GPU training trajectories. Thin lines show individual seeded runs; thick lines show pointwise medians. Curves are neither smoothed nor normalized, and each workload has its own loss scale. The compiled dense panel reuses the same Representax runs against the additional TorchInductor reference.

Figure 4 presents the corresponding TPU trajectories. Several recipes follow similar broad trends while retaining systematic differences in loss level or rate of decrease, notably cross-encoder and image–text training. We retain these differences rather than attribute them to numerical noise without a controlled diagnosis. Comparisons are within a workload and hardware panel, not between GPU and TPU curves with different recipes.

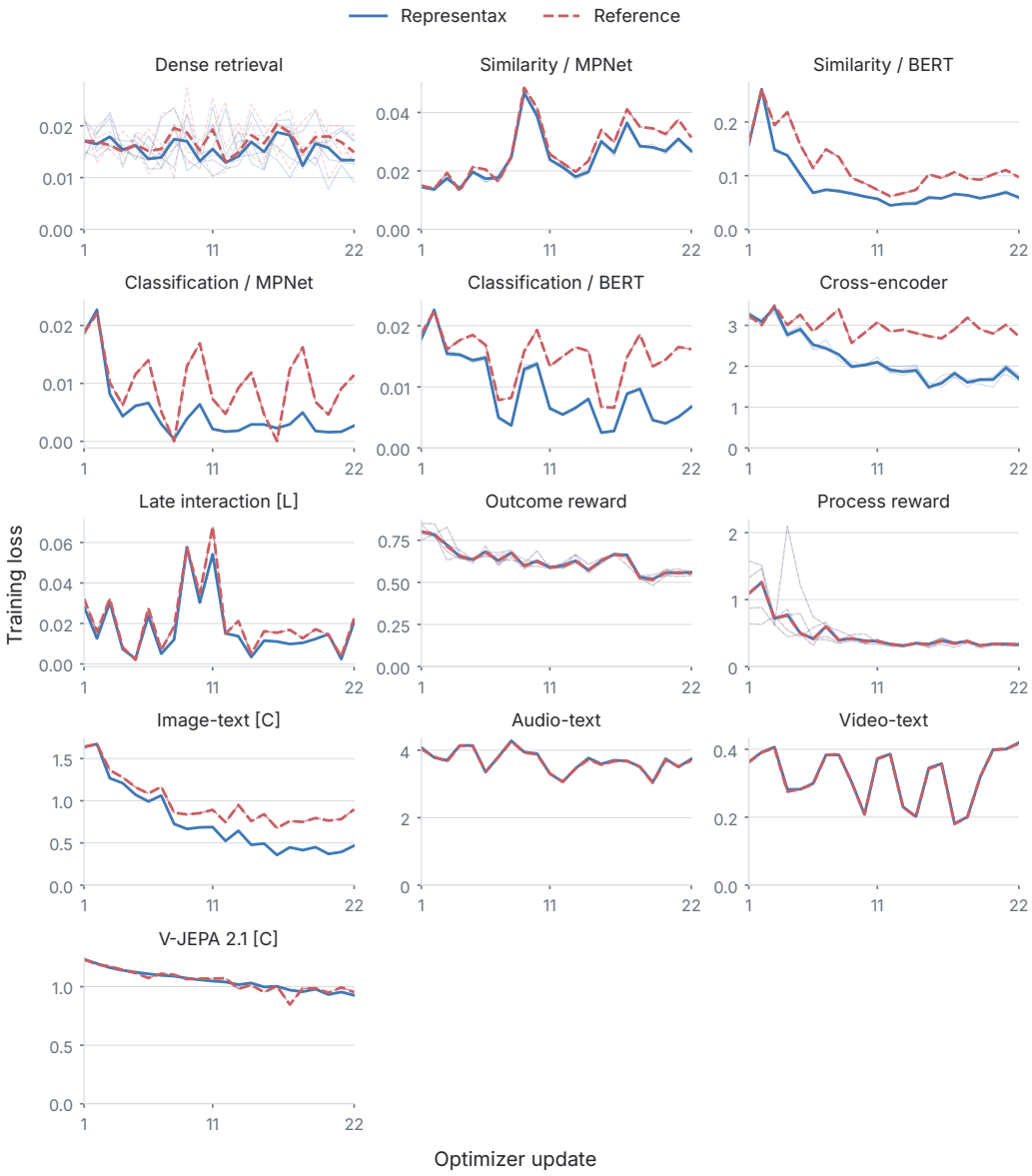


Figure 4: Paired TPU training trajectories on sixteen v5e chips. Thin lines show individual seeded runs; thick lines show pointwise medians, using the same convention as Figure 3. All optimizer updates are shown without smoothing or normalization. Systematic trajectory differences remain visible alongside the throughput results.

B Learning Recipes and Evaluation

B.1 Dense and CLIP Recipes

All adaptation studies report seeds 7, 42, and 773 at the final scheduled checkpoint. Dense adaptation presents each of its 488,576 training pairs once. It uses ETTIN Encoder 150M (149,014,272 parameters), maximum sequence length 128, and buckets 16/96/128. Its custom-VJP query/document chunks are 128/64 and loss-row chunk 64. AdamW uses learning rate 2×10^{-5}, zero weight decay, clipping at one, and cosine decay with approximately 6% warmup. NanoMSMARCO has 50 queries and 5,043 documents; TREC DL 2019 uses 43 queries over 8,841,823 passages. Natural Questions has 3,452 queries and 2,681,468 documents and is a transfer collection, not a training source. Evaluations use the saved corpus/qrel manifests rather than a hand-selected candidate set for each query.

CLIP uses images at 224×224, text length 77, symmetric contrastive loss with scale 20, GradCache chunk eight, and the same peak learning rate and optimizer regularization as dense adaptation. Warmup is 69 of 1,150 updates. The COCO training inventory contains 117,760 images and five captions per image, each presented during the run without repeating an image within a training batch. Flickr30k evaluation has 5,000 text queries and 1,000 image documents; reverse retrieval uses 1,000 image queries and 5,000 text documents. The seed-seven run resumes at update 575 and includes export/reload checks.

B.2 Late-Interaction Regression and Follow-Up

The fourth workload family adapts GTE-ModernColBERT for late-interaction retrieval. It is retained here as a negative result and a direction for further investigation, rather than a positive learning demonstration.

The revised late-interaction recipe uses 452,096 unique query-positive triples with one mined negative per query, global batch 512, query/document caps 32/256, native GradCache chunk eight, temperature 0.02, and AdamW peak learning rate 3×10^{-6}. It runs 1,000 updates, with 60 warmup updates, zero weight decay, and clipping at one. It therefore cycles through more presentations than unique triples. MaxSim evaluation scores are accumulated in FP32. Training loss decreases, but mean NanoMSMARCO nDCG@10 falls from 0.7104 to 0.6925, with final sample SD 0.0010. This regression on the 50-query panel persists across three seeds. It separates objective optimization from held-out improvement (Figure 5).

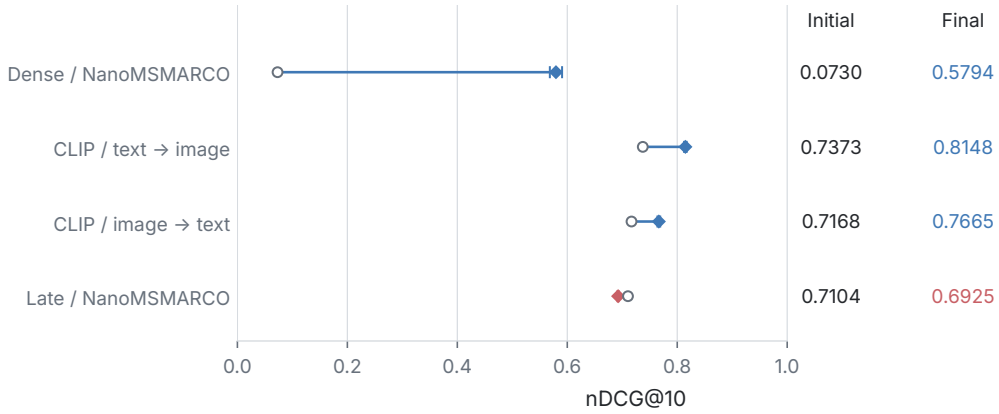


Figure 5: Initial and final nDCG@10 for native learning runs, including the negative late-interaction result. Open circles are initial scores; filled circles and whiskers are final means and sample SD over three seeds. Connecting lines mark endpoints, not interpolated learning curves. All panels share a zero-to-one scale.

The starting checkpoint was distilled, whereas this adaptation uses hard-label contrastive learning without teacher scores. That difference suggests a possible explanation for the regression, but this

Table 9: Text-anchored multimodal adaptation: nDCG@10, mean over three seeds with the sample SD as a superscript. Each strategy is compared with its own initial model. Learning rates and source mixtures differ across strategies.

Strategy	Flickr30k	AudioCaps	MSR-VTT	NanoMSMARCO
Connectors	$+0.1398^{\pm 0.0021}$	$+0.0313^{\pm 0.0011}$	$+0.1041^{\pm 0.0091}$	$+0.0000^{\pm 0.0000}$
Connectors + LoRA	$+0.1743^{\pm 0.0044}$	$+0.0427^{\pm 0.0010}$	$+0.1241^{\pm 0.0014}$	$-0.0387^{\pm 0.0043}$
Full fine-tuning	$+0.1264^{\pm 0.0063}$	$+0.0650^{\pm 0.0027}$	$+0.1002^{\pm 0.0034}$	$-0.0515^{\pm 0.0061}$

Table 10: Prepared text-anchored study data. Counts describe available records, not the number consumed in 2,000 sampled updates. Evaluation is text-to-media, except NanoMSMARCO.

Source / evaluation	Train pairs	Train media	Eval. queries	Eval. documents
COCO / Flickr30k	588,800	117,760	5,000	1,000
AudioCaps	45,178	45,178	4,411	883
MSR-VTT	129,809	7,010	1,000	1,000
MS MARCO / NanoMSMARCO	488,576	—	50	5,043

experiment does not isolate its cause. A useful follow-up would retain per-query rankings, evaluate an additional untouched retrieval panel, and compare objectives while controlling preprocessing and precision. Those evaluations have not been performed; the reported result remains a reproducible small-panel decline, not evidence of a general failure of late-interaction training.

B.3 Multimodal Data and Adaptation

Image, audio, video, and text sources use COCO, AudioCaps, MSR-VTT, and MS MARCO, respectively (Lin et al., 2014; Kim et al., 2019; Xu et al., 2016; Bajaj et al., 2016). Flickr30k supplies held-out image retrieval. Table 9 reports each strategy's absolute scores before and after adaptation; Table 3 in the main text reports changes relative to each run's own initial checkpoint.

Source records are seed-shuffled and sampled in homogeneous batches. Media are decoded through the model processor, with prefetch enabled. Audio is capped at ten seconds; video uses eight frames at 224^2; images use 256^2. Sequence buckets are 128/512/1024, with additional visual-patch and audio-layout buckets recorded in the configuration. Availability filtering means counts need not equal the original dataset release inventories (Table 10).

The learning rates are 2×10^{-4}, 5×10^{-5}, and 10^{-5} for connectors, connectors plus LoRA, and full fine-tuning, respectively. Matryoshka dimensions are 32, 64, 128, 256, 512, and 768. All three strategies use FP32 parameters, BF16 compute, full activation rematerialization, and custom-VJP GradCache chunk two. Earlier capacity probes with larger chunks are not the settings of these final runs. AdamW uses weight decay 0.01, gradient clipping at one, and 120 warmup updates followed by cosine decay. Connector selection includes the vision merger, audio projection, and selected vocabulary delimiter rows. The adapter arm adds rank-eight, alpha-sixteen LoRA across its configured linear modules. The exact name patterns and token rows are preserved in the executable configuration rather than inferred from the strategy name.

Each run presents 64,000 pairs, allocated by its sampled source sequence. Thus the experiment neither covers every available pair nor trains for an equal number of epochs per modality. Evaluations at updates 0, 500, 1,000, 1,500, and 2,000 are retained; the paper reports the last checkpoint. Additional final recall metrics appear in Table 11. These are selected retrieval tasks, not a complete MTEB evaluation (Muennighoff et al., 2022).

Table 11: Final text-to-media or text-to-text recall, mean and sample SD across three seeds. These supplement nDCG; no task-averaged score is formed.

Strategy	Dataset	R@1	R@5	R@10
Connectors	Flickr30k	0.5301 ± 0.0056	0.8210 ± 0.0012	0.8934 ± 0.0002
Connectors	AudioCaps	0.2401 ± 0.0030	0.5792 ± 0.0018	0.7374 ± 0.0043
Connectors	MSR-VTT	0.2963 ± 0.0150	0.5720 ± 0.0151	0.6853 ± 0.0042
Connectors	NanoMSMARCO	0.4800 ± 0.0000	0.7800 ± 0.0000	0.8400 ± 0.0000
Connectors + LoRA	Flickr30k	0.5817 ± 0.0094	0.8454 ± 0.0033	0.9094 ± 0.0022
Connectors + LoRA	AudioCaps	0.2529 ± 0.0010	0.5954 ± 0.0017	0.7423 ± 0.0012
Connectors + LoRA	MSR-VTT	0.3117 ± 0.0061	0.6000 ± 0.0062	0.7057 ± 0.0061
Connectors + LoRA	NanoMSMARCO	0.4200 ± 0.0200	0.7267 ± 0.0416	0.8333 ± 0.0115
Full fine-tuning	Flickr30k	0.5255 ± 0.0093	0.8009 ± 0.0048	0.8765 ± 0.0039
Full fine-tuning	AudioCaps	0.2768 ± 0.0030	0.6191 ± 0.0051	0.7631 ± 0.0060
Full fine-tuning	MSR-VTT	0.2920 ± 0.0078	0.5577 ± 0.0047	0.6853 ± 0.0067
Full fine-tuning	NanoMSMARCO	0.3800 ± 0.0200	0.7133 ± 0.0115	0.8267 ± 0.0416

Figure 6 shows the scheduled evaluations at updates 0, 500, 1,000, 1,500, and 2,000. Media retrieval improves at every checkpoint for the connector and LoRA recipes and is still rising, though more slowly, at the final one; full fine-tuning first dips on AudioCaps before reaching the highest audio score. The text-retention decline under LoRA and full fine-tuning is already present at update 500, is deepest at update 1,000, and recovers only partially afterwards. Connector-only training leaves the text pathway frozen, so its NanoMSMARCO score does not change.

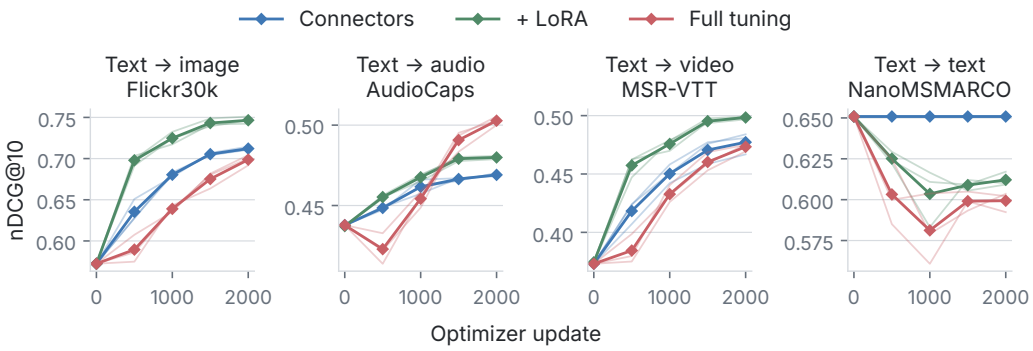


Figure 6: Held-out nDCG@10 at each scheduled evaluation of the multimodal study. Thin lines are seeds; markers and thick lines are means over three seeds. The text pathway is frozen under connector-only training, so its NanoMSMARCO curve is flat.

Table 12: Strong scaling of a fixed 131,072-token update. Means and sample SD use three seeds. Speedup is the mean within-seed ratio; efficiency divides it by GPU count. Memory is the maximum compiled executable footprint per device, not the allocator's reserved pool.

GPUs	Accum.	Tokens/s	Seconds/update	Speedup	Efficiency	GiB
1	16	32015 ± 9	4.094 ± 0.001	1.000	100.0%	32.58
2	8	63000 ± 69	2.081 ± 0.002	1.968	98.4%	32.54
4	4	121971 ± 495	1.075 ± 0.004	3.810	95.2%	32.54
8	2	219975 ± 535	0.596 ± 0.001	6.871	85.9%	33.60

C Scaling Workload Details

Speedup is $S_{n,s} = R_{n,s}/R_{1,s}$ for device count n and seed s; efficiency is $E_{n,s} = S_{n,s}/n$. The reported speedup averages within-seed ratios, not the ratio of independently averaged rates. Compiled executable memory is approximately 32.5–33.6 GiB per device. All 252 updates, including first-use updates, are finite and non-skipped. Parameter-gradient synchronization follows accumulation, and all runs use full rematerialization. Cold compilation takes roughly 126–148 seconds, versus about 14 seconds for cached executable loading. These startup costs are separate from warm throughput.

The controlled ModernBERT encoder uses 32 layers, width 1,280, intermediate width 2,048, twenty attention heads, and vocabulary size 32,768. Global attention occurs every third layer; local attention uses window 128. Global/local RoPE bases are 160,000/10,000 and layer normalization epsilon is 10^{-5}. The MLM head uses exact GELU. Tokenization uses BERT Base Uncased; the larger configured output vocabulary is part of this controlled capacity workload, not a claim of a standard released model.

Global batch is 256 sequences of length 512. All runs use BF16 computation with FP32 parameters and optimizer state; seeds are 7, 42, and 773.

WikiText-2 supplies 23,767 nonempty rows and 2,327,462 corpus tokens. Text is repeated as needed to supply 2,752,512 tokens for 21 updates. The masking procedure selects 15% of eligible tokens with the standard 80/10/10 mask/random/unchanged rule. The random replacement domain is the valid tokenizer vocabulary. This small real-text corpus is adequate for a compute workload but not evidence of general-purpose pretraining.

AdamW uses learning rate 10^{-4}, weight decay 0.01, and clipping at one. Timing includes input preparation and device placement but not offline tokenization. The first update is excluded from warm rates. Memory is derived from the compiled executable's arguments, outputs, temporary buffers, and aliasing, rather than equating JAX's reserved allocator pool with live model memory.

A prior six-RTX-4090 workstation diagnostic showed substantial exposed gradient-collective time. It motivated measurement on the A100 SXM4 node. This is not a controlled NVLink ablation: processors, hosts, and selected microbatches also differ. The paper therefore reports the A100 strong-scaling experiment on its own terms and does not assign all hardware differences to one interconnect feature. FSDP capacity remains unmeasured.

D Executable Task Composition

The following CPU-tested example constructs a symmetric contrastive task with Matryoshka dimensions and differentiates it using ordinary Equinox transformations. The projection and synthetic pairs make the interface self-contained; they are not a benchmark. New tasks can instead implement the scalar-loss interface, and representation-aware tasks can expose the additional boundary used by compatible modifiers and execution paths.

Table 13: Selected upstream model revisions. Abbreviated hashes identify the full revisions preserved in the machine-readable inventory. Names refer to the actual checkpoint, not all models in a family.

Checkpoint	Revision prefix
Sentence Transformers MPNet Base	`e8c3b32edf54`
Google BERT Base Uncased	`86b5e0934494`
MS MARCO MiniLM-L6 cross-encoder	`233902d25c44`
GTE-ModernColBERT-v1	`cbbe53366e56`
Qwen3-0.6B	`c1899de289a0`
Sentence Transformers CLIP ViT-B/32	`327ab6726d33`
LCO-Embedding-Omni-3B-2605	`5f6b5329da51`
ETTIN Encoder 150M	`45d08642849e`
Jina v5 Omni Nano	`b7287f6b6b56`
Official V-JEPA code	`204698b45b37`

```
import equinox as eqx
import jax
import jax.numpy as jnp
from representax.models import DenseEncoder
from representax.tasks import build_task
from representax.tasks.modifiers import MatryoshkaModifierConfig
from representax.tasks.retrieval import (
    MNRConfig, RetrievalConfig, retrieval_batch,
)

model = DenseEncoder(8, 4, key=jax.random.key(0))
x = jnp.eye(8)
batch = retrieval_batch(
    query=x, document=x, positive_mask=jnp.eye(8, dtype=bool),
)
task = build_task(
    RetrievalConfig(), MNRConfig(symmetric=True),
    modifiers=(MatryoshkaModifierConfig(dimensions=(4, 2)),),
)
loss, gradients = eqx.filter_value_and_grad(
    lambda model: task.loss(model, batch).loss
)(model)
```

E Reproduction Inventory

The experiment directories retain launchers and configuration objects for the paired suite (10), dense adaptation (11), late interaction (12), image-text adaptation (13), text-anchored adaptation (14), and MLM scaling (15). Numbering follows the repository's experiment history. The inventory distinguishes a configuration that describes a historical run from a newer configuration intended to improve a future run.

The frozen evidence contains the metric records, run summaries, and source hashes behind every table and figure. A historical-methods inventory captures the 265 paired-run configurations with their environment and data references, plus the longer-run recipes; its source manifest records 693 artifact hashes. Training datasets and checkpoints are obtained through their pinned upstream sources rather than redistributed.

Data revisions include MS MARCO `0d5435254808`, COCO `036f3f8291db`, AudioCaps `b29b3243d6ce`, MSR-VTT `947b139d0c59`, and WikiText `b08601e04326`. Full identifiers, hashes of prepared files, availability filters, and evaluation manifests accompany the records. These preserve the chosen distributions, not a guarantee that upstream media remain downloadable indefinitely.

To rerun training, `experiments/setup.sh` prepares the locked native and reference environments; numbered `run.py` launchers define each experiment. Historical environment records and

source revisions remain authoritative for the reported runs. Repetition also requires suitable accelerators, upstream data and checkpoint access, and the recorded preparation steps.

Table 14: Single-seed ModernBERT execution diagnostics. Each process runs 30 updates; the recorded warm subset contains 13–15 updates. Rates use those subsets, not equal-length wall-clock trials. ST denotes Sentence Transformers. First-use includes compilation or cache loading plus execution; a dash means unrecorded, not zero. The optimized rows combine several changes and are not a single-factor ablation.

Configuration	Warm updates	Examples/s	First-use (s)
Representax / coarse buckets	15	361.08	27.85
Representax / fine buckets	13	393.16	92.04
ST eager / dynamic padding	15	391.69	—
ST eager / fixed padding	15	346.35	—
ST Inductor / dynamic padding	15	667.90	—
ST Inductor / fixed padding	15	583.96	—
Representax / rematerialized	15	341.31	27.28
Representax / optimized, cold	14	618.67	287.51
Representax / optimized, cached	14	620.02	21.60

F Design Analysis

Explicit execution choices help explain performance differences. Earlier ModernBERT diagnostics examine padding, replay, and compilation-cache reuse. These single-seed, 30-update studies use duplicate-heavy data and support systems analysis, not representation-quality claims.

F.1 Diagnostic Setup

These diagnostics come from the earlier ModernBERT dense-retrieval experiment, not the five-seed MPNet panel. Each uses seed 17, batch 128, BF16 compute, FP32 parameters, and 30 updates on one RTX 4090. They retain the same pinned ModernBERT checkpoint and duplicate-heavy source used in that development study; no representation-quality claim is drawn from it. The rate window is the run's recorded warm subset, with 13–15 updates depending on first-use events. Table 14 reports those counts explicitly. They are short diagnostics, not statistically replicated estimates of an optimum.

F.2 Execution Choices

Padding. The shape controls hold the source examples and learning objective fixed while changing padding. Coarse native buckets reach 361.08 examples/s against eager Sentence Transformers' 391.69. Dynamic reference padding executes fewer token positions; matching padded shapes reduces the reference to 346.35, reversing the comparison. Finer native buckets reach 393.16; the compiled reference remains faster at 667.90 and wins under both shape policies.
Replay and startup. Custom-VJP gradient caching reaches 361.08 examples/s against rematerialized GradCache's 341.31. Cumulative changes to layout, replay chunking, RoPE reuse, and static layer execution reach 618.67, without isolating individual effects. Cache reuse reduces compilation plus first execution from 287.51 to 21.60 seconds at similar warm throughput (620.02 examples/s). Repeated-step and startup costs therefore need separate measurement, with their execution settings recorded. Compilation plus first execution includes both activities, rather than measuring pure compilation in isolation. The cold and cache-warm runs have final losses of 4.1371 and 4.1576; this comparison concerns execution cost, not bitwise trajectory equivalence (Figure 7).

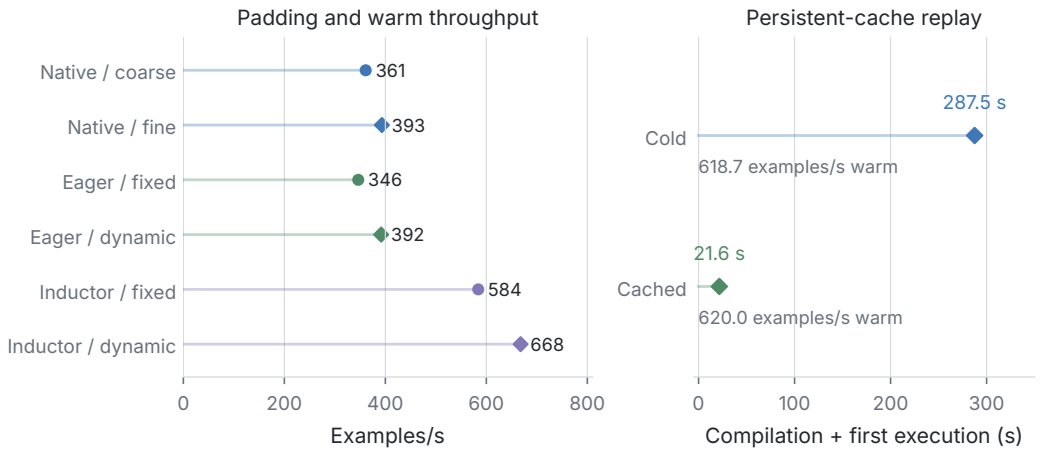


Figure 7: Padding and startup are different execution costs. Left: recorded warm throughput for the coarse/fine native and fixed/dynamic reference padding controls. Each point is one 30-update diagnostic, not a seed average. Right: the final optimized native configuration and a fresh-process cache replay have similar warm rates but different compilation-plus-first-execution costs. Missing reference startup measurements are not plotted as zero.

G Capability and Evidence Boundaries

The experiments exercise complete training and evaluation, four-worker TPU execution, and single-node GPU scaling. The following inventory distinguishes these measurements from broader implementation support. It is not a ranking of framework breadth or a promise that arbitrary combinations have been validated. Precision, objective reduction, stochastic replay, and distributed negative pools require checks for each supported path.

H Learning Signals and Extension Directions

Learning signals shape what an encoding is encouraged to retain, discard, or predict. Table 16 maps the approaches motivating this toolkit. These categories overlap: masked prediction can reconstruct an input, a latent predictor can use a moving teacher, and supervised pairs can define a contrastive objective. The table is a research map, not a taxonomy of mutually exclusive algorithms or a claim that all listed routes are benchmarked.

The implemented denoising task reconstructs clean token sequences through an autoregressive decoder conditioned on an encoder representation. Offline distillation tasks match teacher embeddings, score margins, or candidate-score distributions. Those components are not evidence that image au-

Table 15: Evidence scope. A capability is not a claim of uniform performance across every supported configuration.

Capability	Evidence in this paper
Task and modality breadth	Retrieval, sentence pairs, reranking, reward, image/audio/video contrastive training, and joint-embedding prediction.
GPU and TPU execution	Thirteen recipe rows per backend; qualifications are retained.
Multi-host execution	Four TPU worker VMs, four chips each; coherent JAX and PyTorch/XLA jobs on the same slice.
Adaptation and evaluation	Dense, CLIP, and three text-anchored recipes; final held-out scores and the negative late-interaction result.
Training lifecycle	Saved checkpoints and exports; the CLIP study includes a midpoint resume and export/reload checks.
Multi-GPU scaling	Three-seed 1/2/4/8-GPU replicated-DDP experiment on one A100 node.
Not established	Multi-node GPU scaling, FSDP capacity gains, production reliability, universal speedups, or state-of-the-art quality.

Table 16: Learning signals, intended representational effects, and current scope. Effects are motivations, not guaranteed properties of a learned model.

Approach	Training signal	What it encourages	Representax scope
Supervised learning	Labels, relevance judgments, or preferences	Features useful for the specified task	Classification, ranking, and reward tasks; selected paired measurements.
Reconstruction	Recover an observation from its encoding	Information needed to recover the chosen target	Denoising text autoencoder implemented; not benchmarked here.
Contrastive learning	Relate selected positives and distinguish alternatives	Geometry reflecting the chosen relationships	Dense, token-level, and multimodal objectives; throughput and adaptation studies.
Masked prediction	Infer missing content from context	Dependencies within observations	ModernBERT MLM implemented and used for scaling; broader modality recipes remain open.
Generative prediction	Predict future tokens or observations	Features supporting conditional prediction	Autoregressive decoder used for text reconstruction; general forecasting and world-model recipes are future work.
Latent prediction	Predict a target representation	Predictable structure without requiring raw-input recovery	Joint-embedding predictive tasks; V-JEPA timing measurements.
Self-distillation	Match a teacher or another view's outputs	Consistency across model states or transformed views	Offline teacher-target tasks and JEPA target updates exist; broader online self-distillation recipes are future work.

toencoders, diffusion training, arbitrary forecasting, or complete online self-distillation systems have been implemented or evaluated. Extending the toolkit in those directions requires the corresponding models, data transformations, and task-appropriate evaluation, not merely another loss name.

World-model and latent-diffusion research (Ha & Schmidhuber, 2018; Rombach et al., 2022) motivates studying representations as components of generative and decision-making systems. Additional modalities, including olfactory measurements and asynchronous sensor streams, motivate different encoders and alignment assumptions. The transferable design idea is to expose those scientific decisions while reusing data access, optimization, state management, and distributed execution. Whether a proposed representation helps must be established experimentally.