Value-Based Pre-Training With Downstream Feedback: Meta-Learn Your Pre-Training Objectives

Shuqi Ke
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213, USA
shuqik@andrew.cmu.edu

Giulia Fanti
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213, USA
gfanti@andrew.cmu.edu

Abstract

Continued pretraining is typically optimized with a fixed self-supervised task, yet selected and justified by downstream performance. This creates a coarse feedback loop: practitioners evaluate checkpoints, revise data mixtures or objectives, and restart pretraining runs, while individual pretraining updates remain blind to whether they help the capabilities of interest. We ask whether a small set of verifiable downstream examples can provide step-level feedback during continued pretraining without direct learner supervision. We introduce V-pretraining, which separates a *learner* trained only by a self-supervised loss from a lightweight *task designer* that constructs targets or views for unlabeled batches. Given the current learner and an unlabeled batch, V-pretraining estimates the downstream value of a candidate target or view construction. It does this by predicting the first-order drop in downstream loss caused by a self-supervised update. The designer is trained to increase this value; the learner then applies the resulting self-supervised update, with targets or views detached, so downstream labels never directly update learner parameters. We instantiate V-pretraining to learn adaptive top-K soft targets for language modeling and learned views or masks for self-supervised vision. We observe that V-pretraining can significantly improve target capabilities without harming generalization on vision and language modalities. For instance, under wall-clock-matched continued pretraining, V-pretraining improves GSM8K Pass@1 for Qwen models using 1,024 GSM8K examples only as feedback, including a $+7.4$ point single-run gain for Qwen2.5-0.5B. In vision, V-pretraining improves DINOv3 transfer to ADE20K semantic segmentation and NYUv2 depth estimation while preserving ImageNet linear accuracy, indicating that feedback-guided task construction improves target downstream capabilities without collapsing general-purpose representations.

1 Introduction

Continued pretraining is a standard way to adapt foundation models to new domains and capabilities [27, 26, 49]. For instance, a language model may be further trained on mathematical text to improve reasoning [39], or a visual backbone may be further trained with self-supervised objectives to improve dense prediction [69]. In both cases, the training signal is usually a fixed proxy task [36]: next-token prediction for text [9, 47, 78], reconstruction/view-based or joint-embedding objectives for images [13, 2, 69]. These proxy tasks scale because they do not require dense human annotation [28, 9]. Yet a pretraining run is not judged by the proxy loss itself. It is judged by downstream performance.

This creates a mismatch between the unit of optimization and the unit of selection. Optimization happens one self-supervised update at a time; selection happens after entire continued-pretraining runs are evaluated downstream [44]. In current practice, downstream feedback enters through a

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

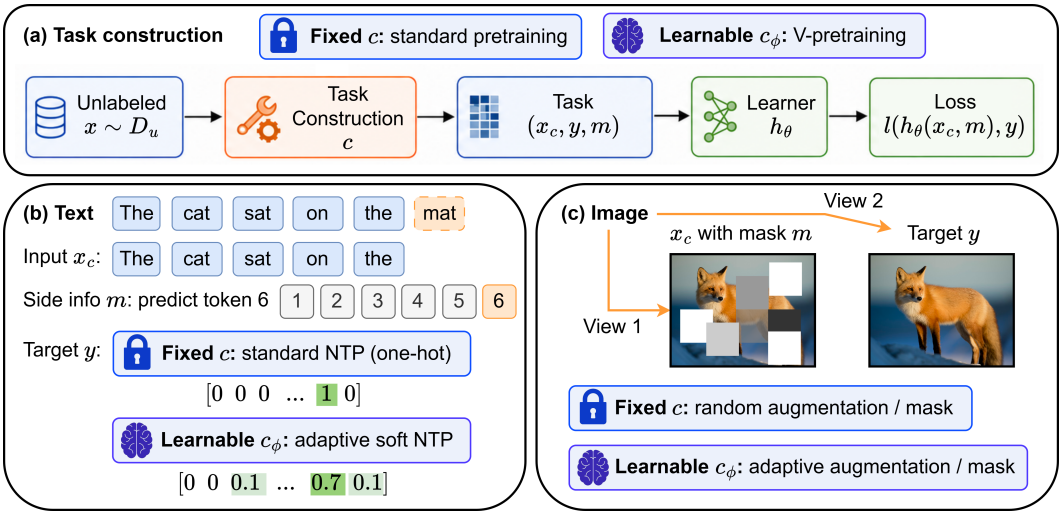


Figure 1: Task construction as the control surface in continued pretraining. A construction rule c maps each unlabeled example $x \sim \mathcal{D}_u$ into a self-supervised prediction problem (x_c, y, m), such as one-hot next-token prediction in language modeling or paired views and targets in DINO-style vision SSL. Standard continued pretraining fixes this rule before training, whereas V-pretraining replaces it with a feedback-trained designer c_ϕ while keeping the learner update self-supervised.

coarse outer loop: practitioners evaluate checkpoints, revise the corpus, objective, curriculum, or augmentation recipe, and launch another run. This makes task-recipe design an expensive black-box search over full pretraining trajectories.

This motivates our question of interest: given a pretrained foundation model, an unlabeled stream for continued pretraining, and a small set of verifiable examples that specify desired downstream capabilities, can we use this feedback during continued pretraining to improve desired capabilities under a limited compute budget, while preserving the generalization capabilities of the pretrained model?

A natural first-pass solution is to fine-tune on the feedback examples. However, fine-tuning is not necessarily the most efficient use of downstream samples when the feedback set is small. SFT [75], preference optimization [59], and RL [50, 67, 1] use downstream examples as learner supervision: labels, preferences, or rewards directly define the learner update. These methods are powerful and complementary, but they answer a different question: how to update the learner on downstream supervision. They do not tell us which unlabeled self-supervised pretraining update was worth taking during a continued-pretraining run; they replace that update with a downstream supervised or reward-based one. Our goal is instead to use downstream examples as a value signal over candidate pretraining updates, without applying downstream gradients or rewards as learner updates.

We introduce **V-pretraining: value-based continued pretraining with downstream feedback**, a framework for the downstream-guided design of self-supervised pretraining tasks (Figure 1). V-pretraining complements the target learner, with parameters θ, with a lightweight task designer, with parameters ϕ. The learner is trained only by a self-supervised pretraining loss. The designer instead dynamically controls how the self-supervised task is constructed during continued pretraining. For example, in language modeling, it shapes the next-token target distribution over a small candidate set; in vision, it selects instance-wise views or masks for self-supervised learning. A feedback batch defines a downstream evaluator gradient, but that gradient is not used to update the learner's parameters directly; rather, it is used only to train the task designer.

The primary technical challenge is that the ideal designer objective is long-horizon and bilevel: minimize a downstream task loss function L_{down}, i.e. choose task constructions whose induced learner trajectory performs well downstream [40, 21]. This is computationally infeasible at pretraining scale. So V-pretraining replaces this objective with a one-step value estimate [34, 55]. If a proposed pretraining task induces gradient $g_{\text{pre}}(\theta; \phi)$ and the feedback set induces downstream gradient

$g_{\mathrm{down}}(\theta)$, then for a small enough $\eta > 0$

$$L_{\mathrm{down}}(\theta - \eta g_{\mathrm{pre}}) \approx L_{\mathrm{down}}(\theta) - \eta g_{\mathrm{down}}^{\top} g_{\mathrm{pre}}. \tag{1}$$

Thus $g_{\mathrm{down}}^{\top} g_{\mathrm{pre}}$ estimates whether the unlabeled update would reduce downstream loss. V-pretraining trains the designer to maximize this value, then detaches the resulting targets or views and updates the learner with the usual pretraining loss. The pretraining loss has a target selected based on the output of the task designer. We instantiate V-pretraining in both language and vision modalities. For language models, the designer learns adaptive top-K soft targets for continued next-token pretraining. For vision backbones, it learns instance-wise views for DINO-style self-supervised learning. In both cases, V-pretraining changes the task construction, not the learner architecture, optimizer, unlabeled stream, or downstream evaluation protocol.

Contributions. We make the following contributions.

- **Conceptual:** We propose V-pretraining, a novel framework for continued pretraining. Notably, we formulate downstream-guided continued pretraining as learnable task construction, which introduces a new channel for guidance during continued pretraining. This guidance comes without directly supervising the pretraining task, but implicitly injects supervision via the task designer.
- **Algorithmic:** We propose a step-level objective for V-pretraining that can be easily and efficiently optimized, and instantiated for different modalities. In particular, we instantiate the same principle as adaptive top-K target construction for language pretraining and learned view construction for DINO/I-JEPA style vision SSL. In both cases, feedback changes the self-supervised task, not the learner objective.
- **Empirical:** We conduct extensive compute-matched experiments to demonstrate the utility of V-pretraining. We demonstrate on multiple modalities that V-pretraining improves downstream performance on target tasks without harming generalization. Under wall-clock-matched continued pretraining, V-pretraining achieves a $+7.4$ point gain or 33% relative improvement on GSM8K Pass@1 in language. In vision, V-pretraining improves DINOv3-ViT-L transfer from 51.33 to 52.40 mIoU on ADE20K and reduces NYUv2 RMSE from 0.5752 to 0.5522. Ablations, one-step probes, decontamination, transfer checks, multitask feedback, and overhead measurements show that the gains require task-relevant feedback and are not explained by generic smoothing, self-distillation, contamination, or direct post-training feedback.

2 V-pretraining Framework

Self-supervised pretraining and continued pretraining do not only specify a loss; they choose how an unlabeled sample is turned into a prediction problem [36]. We call this choice the *task construction rule*. Given an unlabeled example $x \sim \mathcal{D}_u$ from the pretraining data distribution $\mathcal{D}_u$, a (possibly randomized) task construction rule $c(\cdot)$ produces a modified input x_c, a target y, and optional side information m, i.e., $(x_c, y, m) \sim c(x)$. The learner predicts y by outputting $\hat{y} = h_\theta(x_c, m)$ and minimizes

$$L_{\mathrm{pre}}(\theta; c) = \mathbb{E}_{x \sim \mathcal{D}_u} \mathbb{E}_{(x_c, y, m) \sim c(x)} \ell(h_\theta(x_c, m), y). \tag{2}$$

In next-token pretraining [9], given a token sequence $x = (w_1, \ldots, w_n)$, c selects a position $t \in \{1, \ldots, n\}$, sets $x_c = w_{<t}$, uses side information $m = t$, and sets the target to the one-hot next token $y = \delta_{w_t}$. In vision masked modeling [28, 77, 2], c chooses which tokens or patches to hide and defines the reconstruction targets. In DINO-style [69] vision SSL, c samples crops [11], augmentations, and masks; the target is the teacher representation induced by the paired view, and the learner predicts it from the student view.

Standard continued pretraining fixes c before the run. This does not mean the constructed tasks are deterministic: token positions, masks, crops, and augmentations can still be random [12, 28]. It means the distribution that constructs inputs and targets is chosen in advance and does not depend on downstream feedback during training. Our language baseline fixes c to one-hot next-token prediction on the continued-pretraining text stream. Our vision baseline fixes c to the DINO view-generation pipeline. Downstream performance may motivate these recipes, but once training starts, each learner update is optimized for the fixed proxy task rather than for an explicit estimate of downstream value.

V-pretraining alters the construction rule c and define c as a function of the unlabeled data batch (Figure 1). It keeps the learner architecture, optimizer, loss family, and unlabeled data stream fixed,

but replaces the fixed rule c with a designer-controlled rule c_ϕ. In natural language, c_ϕ changes the target distribution for next-token prediction while keeping the context and text stream fixed. In vision, c_ϕ changes instance-wise views or masks while keeping the SSL objective fixed. Thus the controlled object is not the learner itself, but the self-supervised task that generates the learner's next update.

2.1 Indirect feedback through task construction

Let $\mathcal{D}_{\mathrm{fb}}$ be a small feedback set with labels, rewards, or other verifiable downstream signals, and let $\mathcal{D}_{\mathrm{eval}}$ be a held-out evaluation set. Direct post-training methods use $\mathcal{D}_{\mathrm{fb}}$ as learner supervision: SFT [50, 75], preference optimization [42, 59], or RL [1, 67] supply a downstream loss or reward update to θ. V-pretraining uses the same kind of information during pretraining phase. The feedback set may train the task designer but we do *not* use it to directly train the learner.

Formally, every learner update must remain a self-supervised pretraining update on an unlabeled batch $B_t^u \subset \mathcal{D}_u$:

$$g_t^{\mathrm{learn}} = \nabla_\theta L_{\mathrm{pre}}\big(\theta_t; \mathrm{sg}(c_{\phi_t}(B_t^u))\big), \qquad \theta_{t+1} = \theta_t - \eta_t g_t^{\mathrm{learn}}. \tag{3}$$

Here $\mathrm{sg}(\cdot)$ denotes stop-gradient through the constructed targets or views. We write parameter updates using gradient descent for clarity; other optimizers like AdamW [38] can simply replace $-\eta_t g_t$ with the adaptive optimizer step, and all comparisons use the same learner optimizer and schedule. The key invariant is the gradient supplied to the learner: no term of the form $\nabla_\theta L_{\mathrm{down}}(\theta_t; \mathcal{D}_{\mathrm{fb}})$ is applied to θ.

This distinction is essential. Fine-tuning answers how the learner should fit the feedback examples [75]. V-pretraining asks which unlabeled self-supervised update is worth taking before the run is over (Figure 1). The feedback signal is therefore direct-to-designer but indirect-to-learner: it can change $c_\phi(B_t^u)$, and hence the pretraining gradient, but it cannot replace that gradient with a downstream supervised or reward gradient.

2.2 A local value objective for task design

The ideal designer would choose a construction rule whose full continued-pretraining trajectory performs well downstream:

$$\min_\phi L_{\mathrm{down}}(\theta_T(\phi)), \tag{4}$$

$$\theta_{t+1}(\phi) = \theta_t(\phi) - \eta_t \nabla_\theta L_{\mathrm{pre}}\big(\theta_t(\phi); c_\phi(B_t^u)\big), \quad t = 0, \ldots, T-1. \tag{5}$$

Differentiating through this trajectory is impractical at pretraining scale [20, 61, 21, 31]. V-pretraining instead asks a local question at each step: among the tasks that can be constructed from the current unlabeled batch, which one induces a learner gradient that would most decrease the feedback loss? For an unlabeled batch B^u and a labeled feedback batch B^f, the designer induces a candidate pretraining gradient

$$g_{\mathrm{pre}}(\theta; \phi, B^u) = \nabla_\theta L_{\mathrm{pre}}\big(\theta; c_\phi(B^u)\big), \tag{6}$$

and the feedback batch defines an evaluator gradient

$$g_{\mathrm{down}}(\theta; B^f) = \nabla_\theta L_{\mathrm{down}}(\theta; B^f). \tag{7}$$

The candidate learner update is $\theta^+ = \theta - \eta g_{\mathrm{pre}}$. A first-order expansion [34, 55] gives

$$L_{\mathrm{down}}(\theta^+; B^f) \approx L_{\mathrm{down}}(\theta; B^f) - \eta g_{\mathrm{down}}(\theta; B^f)^\top g_{\mathrm{pre}}(\theta; \phi, B^u). \tag{8}$$

We therefore define the value of the proposed pretraining update as the inner product

$$V(\phi; \theta, B^u, B^f) = g_{\mathrm{down}}(\theta; B^f)^\top g_{\mathrm{pre}}(\theta; \phi, B^u), \tag{9}$$

and train the designer with $L_{\mathrm{meta}}(\phi) = -V(\phi; \theta, B^u, B^f)$. Gradients through g_{down} are stopped; the meta-gradient changes ϕ according to how its task construction changes the learner gradient. For efficiency, we compute the dot product on a parameter subset S such as the last transformer blocks or task-relevant projection layers. After the designer update, the task is reconstructed and detached, and the learner applies the gradient-descent pretraining update above. Under smoothness, maximizing V maximizes the first-order term in a lower bound on one-step downstream improvement; the proofs and full algorithm are in Appendix A and Appendix B.

2.3 One-step justification

We now state the local guarantee underlying the value objective. The result is not a long-horizon convergence theorem. Instead, it shows that the alignment objective maximizes a first-order lower bound on one-step downstream improvement.

Proposition 2.1 (1-step downstream improvement lower-bound)**.** *Let L_{down} be L-smooth in θ. Let*

$$\theta^+ = \theta - \eta g_{\mathrm{pre}}(\theta;\phi) \tag{10}$$

for step size $\eta > 0$, and define

$$V(\phi;\theta) = g_{\mathrm{down}}(\theta)^\top g_{\mathrm{pre}}(\theta;\phi), \qquad g_{\mathrm{down}}(\theta) = \nabla_\theta L_{\mathrm{down}}(\theta). \tag{11}$$

Then

$$L_{\mathrm{down}}(\theta) - L_{\mathrm{down}}(\theta^+) \geq \eta V(\phi;\theta) - \frac{L\eta^2}{2}\left\|g_{\mathrm{pre}}(\theta;\phi)\right\|_2^2. \tag{12}$$

The proposition explains the role of V. For sufficiently small learner steps, or when the pretraining update norm is controlled, increasing $g_{\mathrm{down}}^\top g_{\mathrm{pre}}$ increases a certified lower bound on one-step downstream loss decrease. Equivalently, define the one-step bilevel objective

$$J(\phi;\theta) = L_{\mathrm{down}}\left(\theta - \eta\nabla_\theta L_{\mathrm{pre}}(\theta;\phi)\right). \tag{13}$$

A Taylor expansion gives $J(\phi;\theta) = L_{\mathrm{down}}(\theta) - \eta V(\phi;\theta) + O(\eta^2)$. Thus maximizing V is equivalent to minimizing the first-order approximation of the one-step downstream objective.

This local result is deliberately modest. It does not claim that every high-value step guarantees monotonic downstream improvement over a long training trajectory. Stochastic gradients, optimizer state, distribution shift between feedback and evaluation, and interactions across steps can all affect the final trajectory. The purpose of the value objective is to provide a scalable online signal for task design. Section 3 evaluates whether it improves downstream performance over full continued-pretraining runs. We leave the detailed proofs to Appendix B.3.

2.4 Instantiations

Language: adaptive top-K soft targets. For a token sequence $x = (w_1, w_2, \ldots)$, standard next-token pretraining uses context $w_{<t}$ and one-hot target $y = \delta_{w_t}$ V-pretraining keeps the context, token positions, and text stream fixed, but lets the designer replace the one-hot target with a bounded soft target over a small candidate set C_t:

$$C_t = \{w_t\} \cup \mathrm{TopK}_{K-1}\big(\mathrm{sg}(p_\theta(\cdot \mid w_{<t})) \setminus \{w_t\}\big). \tag{14}$$

The true next token is always included. The task construction rule c_ϕ outputs a distribution $r_{\phi,t} \in \Delta(C_t)$ and a gating constant $\alpha_{\phi,t} \in [0, \alpha_{\max}]$, producing weights for each element $v \in C_t$:

$$q_{\phi,t}(v) = (1 - \alpha_{\phi,t})\mathbf{1}[v = w_t] + \alpha_{\phi,t} r_{\phi,t}(v), \qquad v \in C_t. \tag{15}$$

The learner minimizes cross-entropy to detached $q_{\phi,t}$ on the continued-pretraining text. Feedback examples such as GSM8K [17] problems define L_{down} for the designer, but they are not inserted as supervised learner examples.

Vision: learned views for self-supervised learning. We take DINO-style [12, 48, 69] continued SSL as an example here. The base pipeline samples views $V_0(x) = \{v_1^0, \ldots, v_M^0\}$ for an unlabeled image. V-pretraining keeps the DINO loss and student–teacher learner fixed, but lets the designer modify selected views through instance-wise masks or view parameters, producing $V_\phi(x)$. The learner uses the same SSL loss as the baseline,

$$L_{\mathrm{pre}}^{\mathrm{vis}}(\theta;\phi,B^u) = \frac{1}{|B^u|}\sum_{x\in B^u} \ell_{\mathrm{SSL}}(\theta,\bar{\theta};V_\phi(x)), \tag{16}$$

where $\bar{\theta}$ is the EMA teacher used to anchor the DINO SSL pipeline. The feedback evaluator uses lightweight downstream task heads like segmentation and depth estimation on top of the backbone,

$$L_{\mathrm{down}}^{\mathrm{vis}} = \omega_{\mathrm{seg}} L_{\mathrm{seg}}(H_{\mathrm{seg}}(F_\theta(x)), y_{\mathrm{seg}}) + \omega_{\mathrm{dep}} L_{\mathrm{dep}}(H_{\mathrm{dep}}(F_\theta(x)), y_{\mathrm{dep}}) + \cdots, \tag{17}$$

Segmentation [83] or depth estimation [46] labels train the task designer through g_{down}; the backbone update remains the detached SSL loss on unlabeled ImageNet [19] images. By simply replacing the task construction rule c, this instantiation also applies to other SSL method backends like I-JEPA [2].

3 Experiments

We evaluate whether downstream feedback can steer continued pretraining through task construction rather than direct learner supervision. The main experiments ask: (i) Does V-pretraining improve target capabilities under a strict compute comparison? (ii) Does feedback steering over-specialize the model to the feedback task and harm broader transfer? (iii) Are the gains caused by leakage or shortcut supervision, where the learner effectively sees benchmark or feedback labels? (iv) Does the gradient-alignment value signal matter, or are the gains explained by generic smoothing, self-distillation, or additional stochasticity? (v) How does indirect feedback compare with direct post-training, and what overhead does it introduce? (vi) Can the same feedback channel scale across model sizes and support multitask downstream feedback?

3.1 Protocol

All main comparisons are matched by **wall-clock training time on the same hardware configuration**. For each model size, modality, and method, training is run for the same elapsed time under the same hardware and software configuration [33, 41, 35]. The initial learner, unlabeled stream, sequence length or crop configuration, optimizer, numerical precision, and data-loading protocol are held fixed. If V-pretraining introduces additional per-step overhead, it must pay that cost within the same elapsed time. Thus the baseline is allowed to complete as many learner updates and process as many unlabeled tokens or images as its faster training loop permits. V-pretraining receives no extra time to compensate for task-designer computation.

Language. We continue pretraining Qwen1.5 [74] and Qwen2.5 [56] base models on NuminaMath-CoT [37] in main experiments. The **baseline** is standard next-token continued pretraining with one-hot targets. **V-pretraining** uses the same learner configuration and text stream, but replaces one-hot targets with designer-shaped top-K targets. The feedback set contains 1,024 GSM8K training examples, used only to compute g_{down} for the designer. Evaluation is GSM8K test Pass@1 with greedy decoding. See Appendix F for more details and hyper-parameters.

Vision. We continue self-supervised training of DINOv3 ViT-B and ViT-L backbones [69] on ImageNet-1K [19] with a DINO-style student-teacher objective. The **baseline** uses the fixed view-generation pipeline of vanilla DINOv3. **V-pretraining** uses learned instance-wise views with the same SSL objective and learner schedule. Dense-task feedback comes from small labeled ADE20K and NYUv2 pools. We report ADE20K mIoU, NYUv2 RMSE, ImageNet linear accuracy, and transfer diagnostics [46, 83, 57].

3.2 V-pretraining improves target capabilities

Table 1 reports the main wall-clock-matched results. In this part, we focus on tasks that are aligned with the capabilities of the small feedback dataset used to train the task designer. In language, V-pretraining improves GSM8K Pass@1 across the tested Qwen models. The largest gain is the Qwen2.5-0.5B single-run result, improving from 22.20 to 29.60 (a **33% gain**). In the replicated Qwen1.5 runs, V-pretraining improves 0.5B, 4B, and 7B models, with the strongest replicated gain at 4B. The gains decrease with model size, suggesting that larger learners already extract more math-relevant signal from standard continued next-token prediction.

In vision, we provide downstream feedback from dense prediction tasks for V-pretraining. V-pretraining improves dense prediction while preserving global recognition (a task that was *not* used to train the task designer). For DINOv3 ViT-B and ViT-L, ADE20K mIoU increases and NYUv2 RMSE decreases; ImageNet linear accuracy also slightly improves. These results support the feasibility claim: downstream feedback can improve continued pretraining through task construction, even though the learner update remains self-supervised and the training time is matched.

3.3 V-pretraining avoids over-specialization

Because the feedback sets are small and task-specific, V-pretraining could improve the target metric by steering the learner too narrowly. We therefore evaluate tasks that are not used as feedback.

Table 2 evaluates whether feedback steering harms tasks not used as feedback. For language, the OMEGA benchmark [72] measures math reasoning under distribution shift, while MMLU benchmark

Modality	Benchmark	Model	Baseline	V-pretraining
Language	MATH Pass@1 ↑	Qwen2.5-0.5B	22.20	**29.6**
	GSM8K Pass@1 ↑	Qwen1.5-0.5B	$19.15_{\pm 1.16}$	$\mathbf{22.67}_{\pm \mathbf{1.05}}$
	GSM8K Pass@1 ↑	Qwen1.5-4B	$56.48_{\pm 1.56}$	$\mathbf{58.98}_{\pm \mathbf{1.03}}$
	GSM8K Pass@1 ↑	Qwen1.5-7B	$65.26_{\pm 1.06}$	$\mathbf{66.17}_{\pm \mathbf{0.63}}$
Vision	NYUv2 RMSE ↓	DINOv3-ViT-B	0.5888	**0.5697**
	NYUv2 RMSE ↓	DINOv3-ViT-L	0.5752	**0.5522**
	NYUv2 RMSE ↓	I-JEPA-ViT-H	0.7835	**0.7797**
	ADE20K mIoU ↑	DINOv3-ViT-B	48.82	**49.68**
	ADE20K mIoU ↑	DINOv3-ViT-L	51.33	**52.47**
	ADE20K mIoU ↑	I-JEPA-ViT-H	28.03	**28.19**
	ImageNet-1K Acc. ↑	DINOv3-ViT-B	80.74	**81.01**
	ImageNet-1K Acc. ↑	DINOv3-ViT-L	84.07	**84.59**
	ImageNet-1K Acc. ↑	I-JEPA-ViT-H	92.40	**92.50**

Table 1: Main target-capability results under matched wall-clock training budgets. Feedback examples train only the task designer; the learner is updated only by a self-supervised loss on the unlabeled stream. Qwen2.5-0.5B is a single-run diagnostic; Qwen1.5 results report mean and standard deviation.

Benchmark	Model	Baseline	V-pretraining
OMEGA Acc. ↑	0.5B	0.65	0.65
OMEGA Acc. ↑	4B	1.44	**1.88**
OMEGA Acc. ↑	7B	**1.52**	1.50
MMLU Acc. ↑	0.5B	**38.08**	35.01
MMLU Acc. ↑	4B	53.32	**53.51**
MMLU Acc. ↑	7B	**58.81**	58.68

Table 2: Language generalization beyond the GSM8K feedback task. V-pretraining does not over-specialize larger models onto a single downstream task, but the 0.5B model shows a generalization drop on MMLU.

Dataset	Protocol	Baseline	V-pretraining
R-Oxford5k mAP ↑	Easy	0.5268	**0.6048**
R-Oxford5k mAP ↑	Medium	0.4072	**0.4557**
R-Oxford5k mAP ↑	Hard	**0.0867**	0.0820
R-Paris6k mAP ↑	Easy	0.5433	**0.5973**
R-Paris6k mAP ↑	Medium	0.6332	**0.7005**
R-Paris6k mAP ↑	Hard	0.2208	**0.2509**

Table 3: Vision transfer beyond the dense feedback tasks. We evaluate frozen ViT-L representations on instance retrieval. Dense-task feedback improves most retrieval protocols, with a small decrease on Oxford Hard.

[29] measures broader zero-shot multiple-choice knowledge and reasoning. V-pretraining improves OMEGA for the 4B model, remains similar for 0.5B and 7B, and leaves MMLU nearly unchanged for 4B and 7B. The 0.5B model, however, drops in MMLU. This indicates that V-pretraining is a steering mechanism, not a guarantee of monotonic improvement on every unrelated task; small learners can be more sensitive to narrow feedback.

For vision, we evaluate frozen ViT-L representations on Oxford5k & Paris6k instance retrieval [57], which are not used as feedback tasks. Table 3 shows that dense-task feedback improves most retrieval protocols, including Oxford Easy/Medium and all Paris protocols. Oxford Hard is slightly lower. Overall, dense feedback does not collapse the representation onto merely segmentation or depth, though improvements are not uniform across all retrieval settings.

3.4 Does the task designer create a shortcut for the learner?

A separate concern is that V-pretraining might improve the target metric through a shortcut rather than through better self-supervised updates. There are two possible shortcuts. First, the unlabeled continued-pretraining stream might contain near-duplicates of evaluation examples. Second, the task designer might effectively turn feedback examples into learner supervision. We test the first possibility experimentally and rule out the second by the training channel.

Decontamination. We decontaminate NuminaMath-CoT by removing similar samples of GSM8K and MATH using MinHash LSH and n-gram Jaccard similarity [24, 16]. We provide details of this decontamination in Appendix C.2. We then retrain the Qwen1.5-4B baseline and V-pretraining models under the same learner update budget. Table 4 shows that V-pretraining remains above the baseline after decontamination. The margin is smaller than in the original run, but the result suggests that the main gain is not primarily caused by memorizing benchmark-like examples in the unlabeled stream.

Unlabeled stream	Baseline	V-pretraining
Original NuminaMath-CoT	56.48	**58.98**
Decontaminated NuminaMath-CoT	56.7	**57.5**

Table 4: GSM8K Pass@1 for Qwen1.5-4B before and after removing near-duplicates of GSM8K and MATH from the continued-pretraining stream. Baseline and V-pretraining are compared under the same wall-clock training budget. Decontamination does not remove the advantage of V-pretraining.

Method	GSM8K Pass@1 ↑
Baseline	56.48
Rand-feedback	54.31
Uni-smoothing	54.58
Self-distillation	57.61
V-pretraining	**58.98**

Table 5: Ablations in the Qwen1.5-4B setting. V-pretraining is the only variant that trains the designer with $g_{\text{down}}^{\top} g_{\text{pre}}$.

Vision multitask feedback Method	ADE20K mIoU ↑	NYUv2 RMSE ↓	–
Baseline	48.85	0.4139	–
V-pretraining	**49.49**	**0.4135**	–
Language multitask feedback **Method**	**GSM8K Acc. ↑**	**MMLU Acc. ↑**	**MMBP Acc. ↑**
Baseline	44.28	41.90	4.00
V-pretraining	**45.26**	**42.17**	4.00

Table 6: Multitask feedback under wall-clock matching. Each V-pretraining row is a single run using a combined feedback gradient rather than separately tuned single-task feedback.

No shortcut learner supervision. The second shortcut is ruled out by construction. During a V-pretraining learner step, the learner never receives a gradient of the form $\nabla_\theta L_{\text{down}}(\theta; D_{\text{fb}})$, and feedback examples are not inserted into the learner's pretraining batch. The designer can change the target distribution in language or the views/masks in vision, but these constructed targets or views are detached before the learner update.

3.5 Downstream task signal matters for V-pretraining

The main results could have several alternative explanations, including generic label smoothing, self-distillation, or additional stochasticity. We therefore isolate the role of downstream-aligned value feedback in the Qwen1.5-4B setting. Random feedback replaces g_{down} with random vectors [14]. Uniform top-K smoothing uses soft targets without downstream feedback [23, 54]. Self top-K distillation trains on the learner's own candidate distribution [22]. Table 5 shows that these alternatives do not match V-pretraining under the same wall-clock budget. Random feedback and uniform smoothing fall below the one-hot baseline, and self-distillation improves over baseline but remains below downstream-aligned value feedback. Thus the gain is not explained by soft labels or extra meta-gradient noise; it requires task-relevant feedback.

3.6 V-pretraining as a lower-cost feedback before post-training

We show that V-pretraining can be a low-cost complement to post-training. Direct fine-tuning methods such as SFT [50, 75] or GRPO [67] use downstream labels or rewards as learner updates. V-pretraining uses them only to train the task designer.

We compare V-pretraining with the common direct fine-tuning alternatives under a different shared math data source (Figure 2) to show that the trend is robust to data distribution. SFT and V-pretraining both train on OpenMathReasoning [45]: SFT uses the dataset as direct learner supervision, while V-pretraining uses it as the continued training stream with feedback-guided target construction. GRPO also trains on math questions in OpenMathReasoning. So the direct-feedback baselines and V-pretraining are exposed to the same source of math problems but use the signal through different training channels. This comparison is therefore a data-source-controlled diagnostic.

Figure 2 shows that V-pretraining reaches higher GSM8K Pass@1 than SFT runs with comparable or larger additional GPU hours, and also outperforms GRPO alone in this setting. The SFT+GRPO pipeline achieves the highest Pass@1, but at larger training cost. These results support the intended interpretation: V-pretraining can inject useful math capability during continued training before direct post-training, making it a compute-efficient complement to SFT or RL.

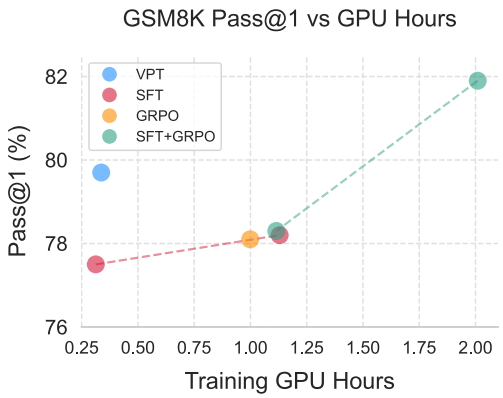


Figure 2: Comparing direct and indirect feedback approaches under a shared math data source. SFT and V-pretraining train on OpenMathReasoning CoT traces, while GRPO trains on OpenMathReasoning questions. V-pretraining uses downstream feedback indirectly through target construction, whereas SFT and GRPO apply supervised or reward feedback directly to the learner. The comparison is diagnostic rather than an apples-to-apples post-training benchmark: SFT+GRPO achieves the highest Pass@1, while V-pretraining provides a lower-cost pre-post-training feedback channel.

Method	**Grad. accum.**	**Tokens/step**	**Step time**	**Tokens/s ↑**	**Peak memory**
Baseline	32	65,536	2.45s	26.7k	31,302 MiB
V-pretraining	20	40,960	1.69s	24.2k	36,440 MiB

Table 7: Runtime profiling for a representative current language run. Both methods use microbatch size 4 and sequence length 512. V-pretraining uses a smaller gradient-accumulation factor to fit the designer and meta-gradient computation, so throughput rather than raw step time is the primary comparison.

Runtime overhead. The preceding comparisons measure downstream performance per reported GPU-hour. We also profile the implementation overhead of the indirect channel itself in a representative language run. This answers a separate question: how much throughput and memory does V-pretraining add relative to standard continued pretraining, before any downstream accuracy is considered?

The V-pretraining run uses a lightweight task designer. The value objective aligns the last two learner layers and performs a meta update every 8 learner optimizer steps. Raw optimizer-step time is not the right cost metric because the meta run uses smaller gradient accumulation to fit the designer and meta-gradient computation. With microbatch size 4 and sequence length 512, the baseline processes $4 \times 32 \times 512 = 65{,}536$ tokens per optimizer step, while V-pretraining processes $4 \times 20 \times 512 = 40{,}960$ tokens per optimizer step.

Table 7 shows that the configured task designer adds measurable cost. Token throughput decreases by 9.4%, and peak memory increases by $5{,}138$ MiB, or 16.4%. The shorter optimizer-step time for V-pretraining is therefore not evidence of lower cost, since each optimizer step processes fewer tokens. This overhead is already charged in our wall-clock-matched main results: the baseline is allowed to process more tokens or learner steps within the same elapsed time. Appendix C reports feedback-set coverage, Pass@k, token-efficiency curves, and additional language diagnostics.

3.7 Capacity scaling and multitask control

Table 1 tested whether the feedback channel transfers across learner scales (measured in number of parameters). A stricter test is whether this channel can compose multiple downstream value signals in one continued-pretraining run. For feedback tasks $j = 1, \ldots, J$, we use a weighted evaluator gradient

$$g_{\text{down}} = \sum_{j=1}^{J} \omega_j g_j, \tag{18}$$

where g_j is the feedback gradient for task j and ω_j controls its relative weight. The designer is then trained to construct self-supervised updates whose learner gradients align with this combined value signal.

For the language multitask run, we use a shared continued-pretraining mixture with three domains: math, code, and general instruction following. Each domain contains a raw component and a high-quality instruction component: OpenWebMath [52] and MetaMathQA [81] for math, codeparrot-clean [6] and Magicoder [76] for code, and C4 [60] and Alpaca [73] for general text (see Appendix E). The mixture assigns 70% probability mass to raw data and 30% to high-quality instruction data, balanced equally across domains. Both the baseline and V-pretraining use the same mixture. Feedback batches are drawn only from the high-quality domain-matched pools and are used to compute the combined evaluator gradient, not to update the learner directly. We present the results in Table 6. V-pretraining improves capabilities for both GSM8K ($44.28 \rightarrow 45.26$) and MMLU ($41.9 \rightarrow 42.17$).

4 Contributions and Limitations

Our contributions can be summarized as follows. (1) We formulate continued pretraining with downstream feedback as task design rather than learner supervision: a small verifier trains a controller that constructs pretraining targets or views, while the learner remains trained only by a self-supervised loss. (2) We derive a scalable step-level value objective, $g_{\mathrm{down}}^{\top} g_{\mathrm{pre}}$, showing that it is the first-order surrogate for one-step downstream improvement and avoids differentiating through full pretraining trajectories. (3) We instantiate the same principle in language and vision, as adaptive top-K target construction for next-token pretraining and learned view construction for self-supervised vision. (4) We provide compute-matched evidence and controls showing that downstream feedback improves value per learner update and that the effect is not explained by random feedback, fixed smoothing, or self-distillation.

V-pretraining provides a local value signal, not a long-horizon optimality guarantee. The alignment objective can be noisy, depends on feedback quality, and may over-steer small learners toward narrow feedback tasks. The method also introduces additional computation for task generation and value-gradient estimation, although our main comparisons match wall-clock time.

Acknowledgments and Disclosure of Funding

This work was supported in part by NSF grant CCF-2338772. This work used the DeltaAI system at the National Center for Supercomputing Applications (award OAC 2320345) [7] through allocation CIS260343 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program [8], which is supported by U.S. National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296. The authors declare no competing interests.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024.

[2] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. *arXiv preprint arXiv:2301.08243*, 2023.

[3] Stephen H. Bach, Bryan He, Alexander Ratner, and Christopher Ré. Learning the structure of generative models without labeled data, 2017.

[4] Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction, 2025.

[5] Wele Gedara Chaminda Bandara, Naman Patel, Ali Gholami, Mehdi Nikkhah, Motilal Agrawal, and Vishal M. Patel. Adamae: Adaptive masking for efficient spatiotemporal learning with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14507–14517, June 2023.

[6] Loubna Ben Allal and Leandro von Werra. codeparrot-clean dataset. `https://huggingface.co/datasets/codeparrot/codeparrot-clean`, 2022.

[7] Brett Bode, Gregory Bauer, Laura Herriott, Volodymyr Kindratenko, and William Gropp. DeltaAI: A national resource for AI/ML research. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, PEARC '25, New York, NY, USA, 2025. Association for Computing Machinery.

[8] Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. ACCESS: Advancing innovation: NSF's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, PEARC '23, pages 173–176, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[10] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeff Wu. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision, 2023.

[11] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9912–9924. Curran Associates, Inc., 2020.

[12] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[14] Jeonghwan Cheon, Sang Wan Lee, and Se-Bum Paik. Pretraining with random noise for fast and robust learning without weight transport. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 13748–13768. Curran Associates, Inc., 2024.

[15] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[16] Karl Cobbe et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[17] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[18] Roi Cohen, Konstantin Dobler, Eden Biran, and Gerard de Melo. I don't know: Explicit modeling of uncertainty with an [idk] token. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 10935–10958. Curran Associates, Inc., 2024.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[21] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning, 2018.

[22] Arvid Frydenlund, Gagandeep Singh, and Frank Rudzicz. Language modelling via learning to rank. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10):10636–10644, Jun. 2022.

[23] Camille Garcin, Maximilien Servajean, Alexis Joly, and Joseph Salmon. Stochastic smoothing of the top-k calibrated hinge loss for deep imbalanced classification. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7208–7222. PMLR, 17–23 Jul 2022.

[24] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[25] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[26] Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats Leon Richter, Quentin Gregory Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. Continual pre-training of large language models: How to re-warm your model? In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023.

[27] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics.

[28] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009, June 2022.

[29] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[30] Dan Hendrycks et al. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[31] Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Convergence analysis and enhanced design, 2021.

[32] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs, 2022.

[33] Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[34] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.

[35] Sameer Kumar, Yu Wang, Cliff Young, James Bradbury, Naveen Kumar, Dehao Chen, and Andy Swing. Exploring the limits of concurrency in ml training on google tpus. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 81–92, 2021.

[36] Yann LeCun. Predictive learning. Invited talk at the 30th Conference on Neural Information Processing Systems (NIPS), December 2016.

[37] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

[38] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[39] Zimu Lu, Aojun Zhou, Ke Wang, Houxing Ren, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. Mathcoder2: Better math reasoning from continued pretraining on model-translated mathematical code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[40] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning (ICML)*, 2015.

[41] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. Mlperf training benchmark. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 336–349, 2020.

[42] Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[43] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In Keh-Yih Su, Jian Su, Janyce Wiebe, and Haizhou Li, editors, *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore, August 2009. Association for Computational Linguistics.

[44] Warren Morningstar, Alex Bijamov, Chris Duvarney, Luke Friedman, Neha Kalibhat, Luyang Liu, Philip Mansfield, Renan Rojas-Gomez, Karan Singhal, Bradley Green, and Sushant Prakash. Augmentations vs algorithms: What works in self-supervised learning, 2024.

[45] Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

[46] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[47] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory

Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[48] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023.

[49] Yixin Ou, Yunzhi Yao, Ningyu Zhang, Hui Jin, Jiacheng Sun, Shumin Deng, Zhenguo Li, and Huajun Chen. How do LLMs acquire new knowledge? a knowledge circuits perspective on continual pre-training. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19889–19913, Vienna, Austria, July 2025. Association for Computational Linguistics.

[50] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[51] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[52] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023.

[53] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4195–4205, October 2023.

[54] Ruotian Peng, Yi Ren, Zhouliang Yu, Weiyang Liu, and Yandong Wen. Simko: Simple pass@k policy optimization, 2025.

[55] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[56] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[57] F. Radenović, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Revisiting oxford and paris: Large-scale image retrieval benchmarking. In *CVPR*, 2018.

[58] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[59] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.

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

[61] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients, 2019.

[62] Neil Rathi and Alec Radford. Shaping capabilities with token-level data filtering, 2026.

[63] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, November 2017.

[64] Colorado J Reed, Sean Metzger, Aravind Srinivas, Trevor Darrell, and Kurt Keutzer. Self-augment: Automatic augmentation policies for self-supervised learning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2673–2682, 2021.

[65] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[66] Chenze Shao, Darren Li, Fandong Meng, and Jie Zhou. Continuous autoregressive language models, 2025.

[67] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[68] Yuge Shi, N Siddharth, Philip HS Torr, and Adam R Kosiorek. Adversarial masking for self-supervised learning. In *International Conference on Machine Learning*, 2022.

[69] Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3, 2025.

[70] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 596–608. Curran Associates, Inc., 2020.

[71] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey, 2022.

[72] Yiyou Sun, Shawn Hu, Georgia Zhou, Ken Zheng, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Omega: Can llms reason outside the box in math? evaluating exploratory, compositional, and transformative generalization, 2025.

[73] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[74] Qwen Team. Introducing qwen1.5, February 2024.

[75] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations (ICLR)*, 2022.

[76] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 52632–52657. PMLR, 21–27 Jul 2024.

[77] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

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

[79] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. *arXiv preprint arXiv:2106.07594*, 2021.

[80] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. Bringing your own view: Graph contrastive learning without prefabricated data augmentations, 2022.

[81] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[82] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[83] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Appendix Contents

A Algorithm **18**

A.1 V-pretraining algorithm 18

A.2 Information bottlenecks 18

B One-step Value Objective **19**

B.1 Derivation 19

B.2 Parameter subset used for alignment 19

B.3 One-step justification and bounds 19

B.4 Relation to the actual optimizer 20

C Additional Language Results **20**

C.1 One-step value probe 20

C.2 Decontamination 21

C.3 Generalization beyond the GSM8K verifier 21

C.4 Feedback-set coverage 21

C.5 Pass@k 21

C.6 Token-efficiency diagnostic 21

C.7 Additional computation cost comparison 22

D Related Work **22**

E Multitask Feedback Data Construction **23**

F Main Language Experiment Setup **25**

F.1 Single-task math continued pretraining 25

F.2 Language implementation branches 26

G Main Vision Experiment Setup **28**

G.1 Baseline continued DINO pretraining 28

G.2 V-pretraining with downstream-aware learned augmentations 28

G.3 Dense evaluation protocol 30

G.4 Hyperparameter selection 30

H Declarations and Impacts **31**

I Compute Resources and Total Compute **31**

Algorithm 1 V-pretraining

Require: Initial learner θ_0, task designer ϕ_0, unlabeled stream $\mathcal{D}_u$, feedback set $\mathcal{D}_{\text{fb}}$, learner budget T, meta-update period r, alignment parameter subset $\mathcal{S}$

for $t = 0, \dots, T-1$ **do**
  Sample unlabeled batch $B_t^u \sim \mathcal{D}_u$.
  if $t \bmod r = 0$ **then**
    Sample feedback batch $B_t^f \sim \mathcal{D}_{\text{fb}}$.
    Construct task $c_{\phi_t}(B_t^u)$ with gradients enabled for ϕ_t.
    Compute $L_{\text{pre}}^{\text{meta}}(\theta_t; \phi_t, B_t^u)$.
    $g_{\text{pre},\mathcal{S}} \leftarrow \nabla_{\theta_{\mathcal{S}}} L_{\text{pre}}^{\text{meta}}(\theta_t; \phi_t, B_t^u)$. ▷ create graph
    $g_{\text{down},\mathcal{S}} \leftarrow \text{sg}\Big(\nabla_{\theta_{\mathcal{S}}} L_{\text{down}}(\theta_t; B_t^f)\Big)$.
    $L_{\text{meta}} \leftarrow -g_{\text{down},\mathcal{S}}^{\top} g_{\text{pre},\mathcal{S}}$.
    $\phi_{t+\frac{1}{2}} \leftarrow \text{Opt}_{\phi}(\phi_t, \nabla_{\phi} L_{\text{meta}})$.
  else
    $\phi_{t+\frac{1}{2}} \leftarrow \phi_t$.
  end if
  Construct task $c_{\phi_{t+\frac{1}{2}}}(B_t^u)$ and detach its targets, masks, or views.
  $L_{\text{pre}}^{\text{learn}} \leftarrow L_{\text{pre}}\Big(\theta_t; \text{sg}\Big(c_{\phi_{t+\frac{1}{2}}}(B_t^u)\Big)\Big)$.
  $\theta_{t+1} \leftarrow \text{Opt}_{\theta}\big(\theta_t, \nabla_{\theta} L_{\text{pre}}^{\text{learn}}\big)$.
  $\phi_{t+1} \leftarrow \phi_{t+\frac{1}{2}}$.
end for

A Algorithm

A.1 V-pretraining algorithm

Algorithm 1 gives the generic V-pretraining procedure. Each iteration has two conceptually separate updates. First, the feedback batch is used to update the task designer through the value objective. Second, the learner is updated only on the resulting pretraining loss over the unlabeled batch. The regularizer $R(\phi)$ is optional and instantiation-specific. It can restrict the designer to stay close to a base pretraining recipe, for example by limiting the amount of probability mass moved away from the true next token [18] or by enforcing mask sparsity and spatial smoothness [68]. These constraints prevent the designer from constructing arbitrary adversarial tasks and keep the learner update anchored to the unlabeled example.

A.2 Information bottlenecks

The algorithm is defined by four information bottlenecks. (1) Feedback labels appear only in the evaluator. The feedback set $\mathcal{D}_{\text{fb}}$ is used to compute L_{down} and g_{down}. These labels or rewards are not inserted into the learner's pretraining batch. (2) The downstream gradient is not a learner update. Although g_{down} is a gradient with respect to learner parameters, it is never passed to the learner optimizer. It is detached and used only as an evaluator vector for updating ϕ. (3) The learner update is self-supervised. The learner step minimizes $L_{\text{pre}}^{\text{learn}}$ on unlabeled data. The targets, masks, or views are produced by the designer but detached before the learner update. Thus the learner receives no supervised downstream loss. (4) Feedback reaches the learner only through task construction. The only path from downstream feedback to θ is

$$\mathcal{D}_{\text{fb}} \to g_{\text{down}} \to \phi \to c_{\phi}(B_u) \to \nabla_{\theta} L_{\text{pre}}. \tag{19}$$

In practice, computing the alignment over all learner parameters can be expensive. We therefore allow the value to be computed on a subset of learner parameters $\mathcal{S}$, such as adapters, the last transformer blocks, or task-relevant projection layers:

$$V_{\mathcal{S}}(\phi; \theta, B_u, B_f) = g_{\text{down},\mathcal{S}}^{\top} g_{\text{pre},\mathcal{S}}. \tag{20}$$

When $\mathcal{S}$ is the full parameter set, this recovers the original V. When $\mathcal{S}$ is restricted, the objective is a computationally cheaper estimator of the same alignment principle. The derivation above is written

for an SGD-style step to expose the basic mechanism. For adaptive optimizers, the same Taylor argument applies to the actual optimizer-induced update direction. Our implementation uses gradient alignment as a lightweight surrogate under a fixed learner optimizer and validates this surrogate empirically with a one-step probe

B One-step Value Objective

B.1 Derivation

At a training iteration, let $B^u \subset \mathcal{D}_u$ be an unlabeled pretraining batch and $B^f \subset \mathcal{D}_{\text{fb}}$ be a feedback batch. The designer induces a candidate pretraining gradient

$$g_{\text{pre}}(\theta; \phi, B^u) = \nabla_\theta L_{\text{pre}}\big(\theta; c_\phi(B^u)\big), \tag{21}$$

and the feedback batch defines an evaluator gradient

$$g_{\text{down}}(\theta; B^f) = \nabla_\theta L_{\text{down}}(\theta; B^f). \tag{22}$$

For the candidate learner update

$$\theta^+ = \theta - \eta g_{\text{pre}}(\theta; \phi, B^u), \tag{23}$$

a first-order expansion of the feedback loss gives

$$L_{\text{down}}(\theta^+; B^f) \approx L_{\text{down}}(\theta; B^f) - \eta\, g_{\text{down}}(\theta; B^f)^\top g_{\text{pre}}(\theta; \phi, B^u). \tag{24}$$

Thus the value score

$$V(\phi; \theta, B^u, B^f) = g_{\text{down}}(\theta; B^f)^\top g_{\text{pre}}(\theta; \phi, B^u) \tag{25}$$

estimates the predicted one-step decrease in downstream loss induced by the proposed self-supervised update. V-pretraining trains the designer by minimizing

$$L_{\text{meta}}(\phi) = -V(\phi; \theta, B^u, B^f) + \lambda R(\phi), \tag{26}$$

where R is an optional regularizer that keeps the constructed task close to the base pretraining recipe.

When updating ϕ, g_{down} is treated as a detached evaluator vector:

$$\nabla_\phi L_{\text{meta}}(\phi) = -\nabla_\phi \left[\text{sg}(g_{\text{down}}(\theta; B^f))^\top \nabla_\theta L_{\text{pre}}(\theta; c_\phi(B^u))\right] + \lambda \nabla_\phi R(\phi). \tag{27}$$

This is a mixed Hessian-vector product. The designer is updated according to how its construction changes the learner gradient, as judged by the downstream evaluator gradient.

B.2 Parameter subset used for alignment

Computing the alignment over all learner parameters can be expensive. We therefore compute the value objective on a subset S of learner parameters:

$$V_S(\phi; \theta, B^u, B^f) = g_{\text{down},S}(\theta; B^f)^\top g_{\text{pre},S}(\theta; \phi, B^u). \tag{28}$$

In language experiments, S is chosen from the final learner blocks or other task-relevant parameters. In the current profiled meta run, the alignment uses the last two learner layers. In vision experiments, S is chosen from the final backbone blocks used by the downstream evaluator heads. This reduces the cost of the mixed derivative while keeping the value signal tied to features used by the feedback tasks.

B.3 One-step justification and bounds

We now state the local guarantee underlying the value objective. The result is not a long-horizon convergence theorem. Instead, it shows that the alignment objective maximizes a first-order lower bound on one-step downstream improvement.

Proposition B.1 (1-step downstream improvement lower-bound)**.** *Let L_{down} be L-smooth in θ. Let*

$$\theta^+ = \theta - \eta g_{\mathrm{pre}}(\theta;\phi) \tag{29}$$

for step size $\eta > 0$, and define

$$V(\phi;\theta) = g_{\mathrm{down}}(\theta)^\top g_{\mathrm{pre}}(\theta;\phi), \qquad g_{\mathrm{down}}(\theta) = \nabla_\theta L_{\mathrm{down}}(\theta). \tag{30}$$

Then

$$L_{\mathrm{down}}(\theta) - L_{\mathrm{down}}(\theta^+) \geq \eta V(\phi;\theta) - \frac{L\eta^2}{2}\left\|g_{\mathrm{pre}}(\theta;\phi)\right\|_2^2. \tag{31}$$

Proof of Proposition B.1. By L-smoothness,

$$L_{\mathrm{down}}(\theta^+) \leq L_{\mathrm{down}}(\theta) + \nabla_\theta L_{\mathrm{down}}(\theta)^\top(\theta^+ - \theta) + \frac{L}{2}\|\theta^+ - \theta\|_2^2. \tag{32}$$

Substituting $\theta^+ - \theta = -\eta g_{\mathrm{pre}}(\theta;\phi)$ gives

$$L_{\mathrm{down}}(\theta^+) \leq L_{\mathrm{down}}(\theta) - \eta g_{\mathrm{down}}(\theta)^\top g_{\mathrm{pre}}(\theta;\phi) + \frac{L\eta^2}{2}\|g_{\mathrm{pre}}(\theta;\phi)\|_2^2. \tag{33}$$

Rearranging yields Equation (31). □

The proposition explains the role of V. For sufficiently small learner steps, or when the pretraining update norm is controlled, increasing $g_{\mathrm{down}}^\top g_{\mathrm{pre}}$ increases a certified lower bound on one-step downstream loss decrease. Equivalently, define the one-step bilevel objective

$$J(\phi;\theta) = L_{\mathrm{down}}\left(\theta - \eta\nabla_\theta L_{\mathrm{pre}}(\theta;\phi)\right). \tag{34}$$

A Taylor expansion gives $J(\phi;\theta) = L_{\mathrm{down}}(\theta) - \eta V(\phi;\theta) + O(\eta^2)$. Thus maximizing V is equivalent to minimizing the first-order approximation of the one-step downstream objective.

This local result is deliberately modest. It does not claim that every high-value step guarantees monotonic downstream improvement over a long training trajectory. Stochastic gradients, optimizer state, distribution shift between feedback and evaluation, and interactions across steps can all affect the final trajectory. The purpose of the value objective is to provide a scalable online signal for task design. Section 3 evaluates whether it improves downstream performance over full continued-pretraining runs.

B.4 Relation to the actual optimizer

The main text and pseudocode use gradient descent for clarity. If the realized learner optimizer update is $d_t(\phi) = \theta^+(\phi) - \theta_t$, the exact first-order value of that update is

$$-g_{\mathrm{down}}(\theta_t)^\top d_t(\phi). \tag{35}$$

For a preconditioned step $d_t(\phi) \approx -\eta_t P_t g_{\mathrm{pre}}(\theta_t;\phi)$, this value becomes $g_{\mathrm{down}}(\theta_t)^\top P_t g_{\mathrm{pre}}(\theta_t;\phi)$. Our implementation uses the unpreconditioned alignment $g_{\mathrm{down}}^\top g_{\mathrm{pre}}$ as a lightweight surrogate under a fixed learner optimizer and schedule. The one-step probe in Appendix C.1 empirically checks that this surrogate is positively correlated with realized downstream loss decrease in the language setting.

C Additional Language Results

C.1 One-step value probe

The value objective predicts the one-step downstream loss decrease as $\eta g_{\mathrm{down}}^\top g_{\mathrm{pre}}$. To test whether this estimate is informative, we compute the predicted improvement on held-out GSM8K probe batches and compare it with the realized decrease in probe loss after a gradient-descent update on g_{pre}. Across probe measurements, the predicted and realized improvements have Pearson correlation $r = 0.657$. The correlation is not expected to be perfect because the estimate is local, minibatch-based, and computed under a fixed surrogate for the realized optimizer step. Its positive value supports using the alignment score as an online training signal for the designer.

C.2 Decontamination

To test whether the language gains are explained by near-duplicates in the continued-pretraining stream, we decontaminate NuminaMath-CoT by removing examples similar to GSM8K and MATH. We use MinHash LSH and n-gram Jaccard similarity to identify candidate overlaps, then retrain the Qwen1.5-4B baseline and V-pretraining models under the same wall-clock budget. Table 8 shows that V-pretraining remains above the baseline after decontamination, though with a smaller margin than on the original stream.

Unlabeled stream	Baseline	V-pretraining
Original NuminaMath-CoT	56.48	58.98
Decontaminated NuminaMath-CoT	56.70	57.50

Table 8: GSM8K Pass@1 for Qwen1.5-4B before and after removing near-duplicates of GSM8K and MATH from the continued-pretraining stream.

C.3 Generalization beyond the GSM8K verifier

Table 9 evaluates whether GSM8K feedback collapses the learner onto the verifier task. V-pretraining improves OMEGA for the 4B learner and leaves MMLU nearly unchanged for the 4B and 7B learners. The 0.5B learner drops on MMLU, indicating that small models can be more sensitive to narrow feedback. These results support the interpretation of V-pretraining as a steering mechanism rather than a uniform improvement guarantee.

Benchmark	Model	Baseline	V-pretraining
OMEGA Acc. ↑	Qwen1.5-0.5B	0.65	0.65
OMEGA Acc. ↑	Qwen1.5-4B	1.44	1.88
OMEGA Acc. ↑	Qwen1.5-7B	1.52	1.50
MMLU Acc. ↑	Qwen1.5-0.5B	38.08	35.01
MMLU Acc. ↑	Qwen1.5-4B	53.32	53.51
MMLU Acc. ↑	Qwen1.5-7B	58.81	58.68

Table 9: Language generalization beyond the GSM8K feedback task. Larger learners do not collapse onto the verifier task, while the 0.5B learner shows a drop on MMLU.

C.4 Feedback-set coverage

We vary the number of GSM8K feedback examples used to compute the evaluator gradient, using 1k, 2k, and 3k examples. Increasing feedback coverage improves the stability and strength of the Qwen1.5-4B gains, with diminishing returns after a few thousand examples. This suggests that the verifier set need not be large to be useful, but its quality and coverage still affect the value signal.

C.5 Pass@k

We evaluate Pass@k for $k \in \{1, 2, 4, 8, 16\}$. V-pretraining improves Pass@k across the tested k values and model sizes. This indicates that the designer improves the solution distribution rather than only changing greedy decoding behavior. The result is consistent with the interpretation that value-guided targets modify reasoning-relevant update directions during continued pretraining.

C.6 Token-efficiency diagnostic

The main comparisons are wall-clock matched. We additionally plot GSM8K Pass@1 as a function of unlabeled tokens processed to diagnose update quality. This diagnostic asks whether a V-pretraining update carries more downstream value per unlabeled token once the designer begins steering. In the Qwen1.5-4B setting, V-pretraining reaches 56.18 Pass@1 after 400 learner steps, approximately 1.3×10^7 unlabeled tokens, while the baseline requires roughly 10^3 steps to reach comparable accuracy. The curve shows an early transient dip before the designer stabilizes; this is mitigated by a

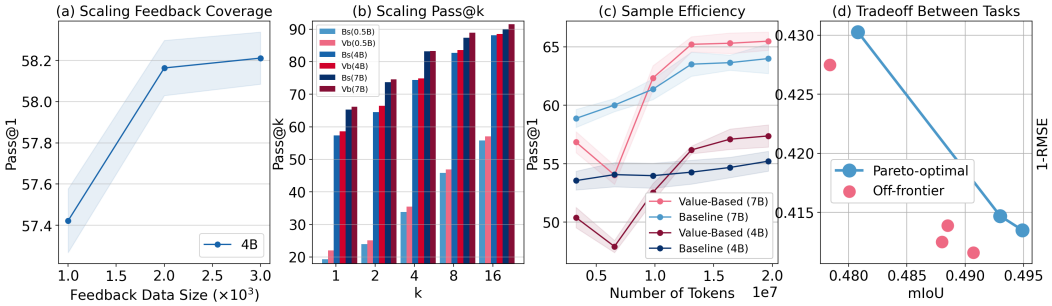


Figure 3: Additional language diagnostics. (a): GSM8K Pass@1 as a function of feedback-set size. (b): Pass@k across tested model sizes. (c): token-efficiency diagnostic plotting Pass@1 against unlabeled tokens processed. These curves are diagnostics, not the primary fairness criterion. (d): Tradeoff between segmentation (mIoU) and depth estimation (1-RMSE) induced by varying feedback and task-designer hyperparameters.

Method	**MATH Pass@1** ↑	**Hardware / time**	**Relative cost**
Baseline	52.48	10 min on RTX 6000 Ada	1×
V-pretraining	**53.92**	10 min on RTX 6000 Ada	1×
SFT+GRPO	53.8	16 hrs on H100	15–25×

Table 10: Direct-vs-indirect feedback diagnostic. GRPO applies downstream reward directly to the learner; V-pretraining applies downstream feedback only to the task designer. The comparison is diagnostic, not a claim that indirect feedback replaces post-training.

burn-in schedule that delays designer updates. This analysis is diagnostic and does not replace the wall-clock-matched results.

C.7 Additional computation cost comparison

Table 10 gives a MATH [30] diagnostic: V-pretraining reaches comparable Pass@1 to an SFT+GRPO pipeline in the reported setting, while using the less training-time budget. Because the stages, hardware, and optimization pipelines differ, this comparison should be read only as evidence that some downstream value can be injected before direct reward optimization.

D Related Work

Positioning. We study *controlled pretraining* under a fixed unlabeled stream and learner update budget. A small feedback set of verifiable downstream tasks provides verified goal information, but it is used only to train a lightweight controller that reshapes the *pretraining target* (or views). The foundation model is *never* updated on downstream labels. This differs from most label-efficient paradigms, which improve performance by creating labels or pseudo-labels and then training the main model on them.

Post-training injects direction late. Supervised fine-tuning and preference optimization steer models by directly updating the foundation model on labeled examples or preferences [15, 51, 58]. These methods are highly effective, but they operate after proxy pretraining has already shaped the representation space. Our approach is complementary: we inject goal information *during* pretraining by shaping the unlabeled training signal rather than updating the learner on downstream labels.

Weak/semi-supervision: scalable supervision by producing labels, not by steering pretraining updates. A broad literature improves *supervision scalability* by learning from imperfect labels or by manufacturing labels from weak sources, spanning weak supervision and data programming [63, 3], distant supervision [43], semi-supervised learning [70], robust learning from noisy labels [71], and more recently weak-to-strong generalization as a way to elicit strong capabilities from weak supervision [10]. Across these settings, progress typically comes from generating (pseudo-)labels and then training the *main model* on task-defined targets, often with repeated inference or teacher–student

refinement that is not compute-matched to pretraining-scale update budgets. Our method can be viewed as a task-agnostic *pretraining analogue* of weak-to-strong generalization: a small feedback set of verifiable downstream tasks provides weak but reliable goal information, yet the foundation model is never trained on downstream labels; instead, the feedback trains a lightweight controller that reshapes the *self-supervised* target/views so that each unlabeled gradient step has higher downstream value.

Directing pretraining without step-level downstream feedback: proxy objectives and view design. Most improvements to foundation-model pretraining change the proxy objective or the view/augmentation pipeline while keeping the training signal fixed: in language this includes next-token-based variants and domain-shaped objectives specified *a priori* [9, 82, 4, 66], while in vision SSL many methods learn from global semantics via contrastive/joint-embedding objectives [13, 25, 12] and others inject spatial structure through handcrafted augmentations or predictive objectives such as masked modeling and JEPA-style prediction [28, 2]. These approaches can yield strong representations, but the *direction* they impose is largely static: the target construction does not adapt online to what a downstream verifier says is valuable for the current model and example [68, 5]. In contrast, value-based pretraining introduces a control loop that uses a small feedback set of verifiable downstream tasks to *modulate the pretraining target/views* so that each unlabeled update aligns with downstream improvement, directly addressing the value-per-step and feedback-efficiency pressures highlighted in our introduction.

Bilevel optimization and influence. Downstream-aware task design naturally leads to bilevel optimization and unrolled differentiation through training [40, 21], which is costly at pretraining horizons. To our knowledge, existing work has not optimized both pretraining tasks and SSL augmentations in a bilevel optimization [64]; the closest approaches use a coordinate-descent-like step-wise optimization [79, 80, 32]. We circumvent the computational challenges of this bilevel optimization using influence-style methods that estimate the effect of training updates on downstream loss from gradients [34, 55]. We build on these approximations but apply them to *target/view construction during pretraining*: a controller learns to reshape the unlabeled supervision signal so that each proxy update aligns with downstream improvement.

E Multitask Feedback Data Construction

Goal. The multitask language experiment is designed to test whether the same indirect feedback channel can steer continued pretraining toward multiple capability domains in a single run. We consider three domains: mathematical reasoning, code generation, and general instruction following / world knowledge. The key design constraint is that feedback should provide a clean value signal for the task designer, while the learner still trains on a mixed continued-pretraining stream.

Pretraining mixture. The final multitask continued-pretraining corpus contains both raw domain text and high-quality instruction-format data. The raw component provides broad next-token prediction coverage; the high-quality component anchors the stream in task-relevant formats. The three domains are:

- **Math:** OpenWebMath [52] as raw mathematical text, and MetaMathQA [81] as high-quality math instruction data.
- **Code:** codeparrot-clean [6] as raw code-domain text, and Magicoder OSS-Instruct-75K [76] as high-quality code instruction data.
- **General:** C4 English [60] as raw web text, and Alpaca [73] as high-quality general instruction data.

The pretraining mixture assigns total probability mass 0.7 to raw data and 0.3 to high-quality instruction data. Within each group, probability mass is balanced equally across the three domains. Thus each raw domain receives mass $0.7/3$, and each high-quality domain receives mass $0.3/3$. Sampling is weighted rather than performed by simple concatenation, so large corpora such as C4 do not dominate smaller instruction datasets.

Why mix raw and high-quality data? We found that instruction-only mixtures make the experiment less diagnostic. If the pretraining stream consists almost entirely of clean instruction pairs

(which is often infeasible at practical pretraining scale), the standard baseline can learn much of the task format directly from the pretraining data. In that case, improvements may reflect direct exposure to instruction data rather than feedback-controlled task construction. Conversely, very small or poorly balanced mixtures can over-repeat small domains, especially code, and create domain-specific overfitting. The final raw/high-quality mixture therefore separates two roles: raw text supplies broad domain coverage, while high-quality examples keep the stream aligned with the capabilities evaluated downstream.

Sampling caps. Before tokenization and chunking, we cap the number of source items drawn from each domain. In the full multitask builder, the default caps are 50k raw source documents per domain and 5k high-quality examples per domain for the pretraining portion. Because raw documents are chunked into fixed-length token sequences after tokenization, the number of resulting training instances can be substantially larger than the number of original source documents. Smaller pilot runs use the same construction logic with reduced source counts.

Sequence construction. All examples are tokenized with maximum sequence length 512. Raw-text examples are treated as standard next-token prediction data: an end-of-sequence token is appended, the token stream is chunked into contiguous sequences of length 512, and the learner predicts every token in the chunk. Very short chunks are discarded.

High-quality instruction examples are converted to a shared instruction-response format. Each example contains an instruction, an optional input field, and a response. The learner loss is masked on the prompt tokens and applied only to the response tokens. This formatting is used for MetaMathQA, Magicoder, and Alpaca so that the three high-quality domains share a common surface structure.

Feedback data. The feedback batches use high-quality instruction-format data only. We use MetaMathQA for math feedback, Magicoder OSS-Instruct-75K for code feedback, and Alpaca for general instruction feedback. These feedback pools are balanced equally across the three domains. For a feedback task j, let g_j denote the evaluator gradient computed from the corresponding feedback batch. The multitask value signal is

$$g_{\text{down}} = \sum_{j=1}^{J} \omega_j g_j, \tag{36}$$

where ω_j controls the relative weight of task j. In the reported multitask run, feedback batches are domain-balanced; the resulting combined gradient trains the task designer to construct pretraining updates that align with multiple downstream-relevant directions.

Separation of roles. The same dataset source can play different roles depending on how it is used. Examples in the continued-pretraining mixture are learner data: they define next-token prediction updates. Feedback examples are evaluator data: they define g_j for the task designer. They are not inserted as learner updates during the meta step. Thus the multitask experiment preserves the indirect-feedback channel: the learner is trained on the continued-pretraining stream, while the feedback sets train only the task designer.

Pilot mixtures. We tested two earlier data constructions before using the final raw/high-quality mixture. The first was a MetaMathQA-heavy blend with small code and MMLU-style components. This made the math pretraining stream too close to GSM8K/MATH-style evaluation and required substantial repetition of small code subsets. The second was an equal-weight instruction-only mixture using filtered math instruction data, Magicoder, and Alpaca. This removed the most obvious over-repetition issue, but the setting remained too clean: both the baseline and V-pretraining could learn strongly from direct instruction-format continuation alone. These pilot results motivated the final mixture, which combines broad raw-domain coverage with a smaller amount of high-quality instruction data.

Evaluation. The multitask run is evaluated on held-out benchmarks that probe the three target capability domains: GSM8K for mathematical reasoning, MMBP/MBPP-style code generation, and MMLU for broad knowledge and instruction following. These benchmarks are used for reporting downstream performance, not as learner-update targets. The purpose of the multitask experiment is

not to show that all metrics improve uniformly, but to test whether a combined feedback gradient can steer one continued-pretraining run toward several capabilities under a fixed compute budget.

F Main Language Experiment Setup

Purpose. The main language experiments test whether a small downstream verifier can steer continued language-model pretraining without directly training the learner on the verifier examples. The target capability is mathematical reasoning. The learner is continued on a math-oriented pretraining stream, while a small GSM8K feedback set is used only to train the task designer through the gradient-alignment objective. The reported metric is GSM8K test Pass@1.

F.1 Single-task math continued pretraining

Learners. The main single-task language results use Qwen-family [74, 56] causal language models as learners. The reported runs include Qwen1.5 learners at multiple scales and a Qwen2.5-0.5B learner. Unless otherwise stated, each method starts from the same initial checkpoint for a given model size and uses the same learner optimizer, learning-rate schedule, numerical precision, sequence length, and wall-clock training budget.

Unlabeled pretraining stream. The single-task math runs use NuminaMath-CoT [37] as the continued-pretraining stream. Each example is formatted as a problem followed by a solution. Training uses causal language modeling on the solution span: prompt tokens are masked in the loss, and the learner predicts only answer or solution tokens. Both the baseline and V-pretraining use the same pretraining stream and the same loss mask.

Baseline. The baseline is standard continued next-token pretraining on the math stream. For a token position t with true next token y_t, the learner is trained with a one-hot target δ_{y_t}:

$$L_{\text{base}}(\theta) = \frac{1}{|B^u|} \sum_{x \in B^u} \frac{1}{|I(x)|} \sum_{t \in I(x)} \text{CE}(\delta_{y_t}, p_\theta(\cdot \mid x_{<t})) \,, \tag{37}$$

where B^u is an unlabeled pretraining batch, CE denotes the cross-entropy loss, and $I(x)$ denotes the solution-token positions included in the learner loss.

Feedback set. The feedback set contains 1,024 GSM8K training examples. These examples define the downstream evaluator loss used to compute g_{down}, but they are not inserted into the learner's pretraining stream and are not used as supervised learner updates. For a feedback batch B^f, the evaluator loss is

$$L_{\text{down}}(\theta; B^f) = \frac{1}{|B^f|} \sum_{(u,a) \in B^f} \frac{1}{|A(u,a)|} \sum_{t \in A(u,a)} -\log p_\theta(a_t \mid u, a_{<t}), \tag{38}$$

where u is the problem statement, a is the verified solution, and $A(u,a)$ denotes answer-token positions. The resulting gradient $g_{\text{down}} = \nabla_\theta L_{\text{down}}(\theta; B^f)$ is detached and used only as an evaluator vector for the task designer.

V-pretraining objective. V-pretraining keeps the same text stream and causal-LM learner but replaces the fixed one-hot next-token target with a designer-shaped soft target over a small candidate set. For each eligible token position t, we form

$$C_t = \{y_t\} \cup \text{TopK}_{K-1}(\text{sg}(p_\theta(\cdot \mid x_{<t})) \setminus \{y_t\}) \,, \tag{39}$$

so the true next token is always included and the remaining candidates are the learner's high-probability alternatives. The task designer outputs a distribution $r_{\phi,t} \in \Delta(C_t)$ and a token-dependent mixing coefficient $\alpha_{\phi,t} \in [0, \alpha_{\max}]$. The learner target is

$$q_{\phi,t}(v) = (1 - \alpha_{\phi,t})\mathbf{1}[v = y_t] + \alpha_{\phi,t} r_{\phi,t}(v), \qquad v \in C_t, \tag{40}$$

and $q_{\phi,t}(v) = 0$ for $v \notin C_t$. Setting $\alpha_{\phi,t} = 0$ recovers the baseline one-hot objective.

During a meta update (designer update), the designer is trained to make the induced pretraining gradient align with the downstream evaluator gradient. Let

$$g_{\mathrm{pre,S}} = \nabla_{\theta_S} L^{\mathrm{LM}}_{\mathrm{pre}}(\theta; \phi, B^u), \qquad g_{\mathrm{down,S}} = \mathrm{sg}\big(\nabla_{\theta_S} L_{\mathrm{down}}(\theta; B^f)\big)\,, \tag{41}$$

where S is the subset of learner parameters used for alignment. The implementation maximizes gradient alignment, using either the dot product from the main derivation or its cosine-normalized version:

$$L_{\mathrm{meta}}(\phi) = -\mathrm{Align}(g_{\mathrm{down,S}}, g_{\mathrm{pre,S}}). \tag{42}$$

After the designer update, the soft targets are recomputed and detached. The learner update is then an ordinary causal-LM pretraining update:

$$g^{\mathrm{learn}} = \nabla_\theta L^{\mathrm{LM}}_{\mathrm{pre}}\big(\theta; \mathrm{sg}(q_\phi), B^u\big), \qquad \theta^+ = \theta - \eta g^{\mathrm{learn}}. \tag{43}$$

Thus GSM8K feedback affects the learner only by changing the self-supervised next-token targets on NuminaMath-CoT examples. No GSM8K supervised gradient is applied to the learner in our main experiments (Table 1).

Evaluation. We evaluate on the GSM8K test set using the same decoding and answer-extraction protocol for all methods in a comparison. Main results report Pass@1. Additional Pass@k, feedback-coverage, and token-efficiency diagnostics are reported in separate appendix sections.

F.2 Language implementation branches

We used two related language implementations for different experiments. They share the same conceptual student–designer structure, but differ in learner parameterization, data regime, and alignment parameters. We report them separately to avoid conflating parameter-efficient adaptation with full-parameter continued pretraining.

LoRA reference implementation. The compact reference implementation uses Qwen2.5-0.5B as the learner and trains LoRA adapters rather than all model parameters. The LoRA configuration is

$$r = 8, \qquad \alpha_{\mathrm{LoRA}} = 16, \qquad \mathrm{dropout} = 0.05.$$

Adapters are applied to the standard attention and MLP projections: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. This branch is therefore a parameter-efficient language adaptation setting.

The LoRA reference branch uses a multitask instruction-style pretraining stream with MathInstruct, Magicoder OSS-Instruct-75K, and Alpaca. MathInstruct is filtered to remove sources derived from GSM8K and MATH. All examples are converted to a shared causal-LM format with prompt masking, so only answer tokens contribute to the learner loss.

The task designer is a compact decoder-only TopKAugmentor. It scores only the student learner's top-K next-token candidates rather than the full vocabulary. It outputs both the top-K distribution $r_{\phi,t}$ and the smoothing gate $\alpha_{\phi,t}$. Gradient alignment is computed on LoRA parameters from the last two transformer layers. The default settings in this branch are:

student learning rate	2×10^{-4},
designer learning rate	1×10^{-4},
batch size	4,
gradient accumulation	8,
maximum steps	2000,
pretraining sequence length	1024,
downstream sequence length	512,
meta sequence length	256,
K	64,
$\alpha_{\max}$	0.5,
meta-update frequency	every 8 learner steps,
alignment subset	last 2 LoRA layers.

Full-parameter implementation. The full-parameter implementation uses Qwen2.5-0.5B with all learner parameters trainable. This branch is a direct full continued-pretraining implementation of the same feedback-controlled task-design idea. The baseline trains the learner with ordinary causal language modeling on the selected pretraining stream. Raw text uses standard next-token prediction, while instruction-format examples mask prompt tokens and train only on response tokens.

The default full-parameter baseline uses AdamW with bf16 learner precision and the following settings:

learning rate	1.5×10^{-5},
batch size	4,
gradient accumulation	32,
effective batch size	128,
epochs	2,
warmup ratio	0.03,
sequence length	512,
weight decay	0,
max gradient norm	1.0,
learner precision	bf16.

The corresponding V-pretraining version uses the same learner and pretraining stream, but adds a small TopKAugmentor. Unless otherwise stated, the augmentor has hidden size 256, 6 layers, 4 attention heads, top-$K = 64$, and $\alpha_{\max} = 0.5$. The augmentor is trained in fp32 even when the learner is trained in bf16. Alignment is computed over a selected subset of full learner parameters, usually the last few transformer layers, controlled by `align_last_n_layers`.

Domain-matched alignment for multitask language runs. In multitask language runs, we compute gradient alignment within matching domains rather than pooling all tasks into a single undifferentiated feedback batch. For a domain $d \in \{\text{math}, \text{code}, \text{general}\}$, the method pairs a same-domain pretraining batch with a same-domain feedback batch:

$$B_d^u \leftrightarrow B_d^f. \tag{44}$$

It then computes

$$g_{\text{pre},d} = \nabla_{\theta_S} L_{\text{pre}}(\theta; \phi, B_d^u), \qquad g_{\text{down},d} = \text{sg}\Big(\nabla_{\theta_S} L_{\text{down}}(\theta; B_d^f)\Big), \tag{45}$$

and trains the designer with the mean alignment objective

$$L_{\text{meta}}(\phi) = -\frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \text{Align}(g_{\text{down},d}, g_{\text{pre},d}). \tag{46}$$

This domain-matched design avoids a failure mode in which a single combined feedback gradient pulls the designer toward a direction inconsistent with the current pretraining domain.

Feedback formatting variants. For multitask language experiments, the default feedback pools are MetaMathQA for math, Magicoder for code, and Alpaca for general instruction following. These feedback examples are formatted consistently with the instruction-style pretraining examples. We also implemented an evaluation-aligned feedback variant, in which feedback examples are reformatted to more closely match downstream benchmark prompts: math feedback is written in a GSM8K-style format, code feedback in an MBPP-style format with markdown fences removed, and knowledge feedback in an MMLU multiple-choice format. This variant is treated as a separate experimental branch rather than the universal default.

Separation of learner and feedback data. Across both implementation branches, the key invariant is unchanged. Pretraining examples define learner updates. Feedback examples define evaluator gradients for the task designer. Even when a feedback dataset has the same source name as a high-quality pretraining dataset, its role is different: feedback batches are not inserted as learner updates during the meta step. The only learner update remains a causal-LM loss on the current pretraining batch with detached designer-shaped targets.

G Main Vision Experiment Setup

Purpose. The vision experiments test whether downstream dense-prediction feedback can steer continued self-supervised pretraining without directly training the visual backbone on dense labels. The learner is a DINOv3 vision transformer [69] continued on unlabeled ImageNet-1K images with a DINO-style self-supervised objective. The feedback tasks are ADE20K semantic segmentation [83] and NYUv2 depth estimation [46]. Their labels are used only to train the task designer through the value objective; the backbone learner is still updated only by a self-supervised loss on unlabeled images.

G.1 Baseline continued DINO pretraining

Learners and unlabeled data. The main vision results use pretrained DINOv3 ViT backbones, including ViT-B and ViT-L (we discuss the I-JEPA setup later). Each run starts from the same pretrained checkpoint for a given backbone size and continues self-supervised training on ImageNet-1K. The same unlabeled image stream, crop configuration, learner optimizer, learning-rate schedule, teacher momentum schedule, numerical precision, and training budget are used for the baseline and V-pretraining comparison.

DINO-style SSL objective. The baseline is standard continued DINO-style self-supervised learning. The student consists of the DINOv3 backbone and a projection head. The teacher is an exponential-moving-average copy of the student. For each image, the fixed augmentation pipeline samples multiple views: two global crops and six local crops by default, with color jitter, grayscale augmentation, Gaussian blur, and solarization [11, 12]. The student is trained to match teacher predictions across non-matching views, with temperature scaling and a running teacher-output center.

Let $V_0(x) = \{v_1^0, \dots, v_M^0\}$ denote the views produced by the fixed DINO augmentation pipeline for an unlabeled image x. The baseline minimizes

$$L_{\mathrm{ssl}}^0(\theta; B^u) = \frac{1}{|B^u|} \sum_{x \in B^u} \ell_{\mathrm{DINO}}\big(\theta, \bar{\theta}; V_0(x)\big)\,, \tag{47}$$

where θ are the student parameters and $\bar{\theta}$ are the EMA teacher parameters. The task construction rule c is fixed before training: the views are stochastic, but their sampling distribution and augmentation recipe do not depend on downstream feedback.

Baseline learner update. The baseline learner update is therefore

$$g_t^{\mathrm{base}} = \nabla_\theta L_{\mathrm{ssl}}^0(\theta_t; B_t^u), \qquad \theta_{t+1} = \theta_t - \eta_t g_t^{\mathrm{base}}. \tag{48}$$

No ADE20K or NYUv2 labels are used in the baseline continued-pretraining update.

G.2 V-pretraining with downstream-aware learned augmentations

Control surface. V-pretraining keeps the DINO learner, SSL objective, unlabeled stream, and optimizer fixed, but replaces part of the fixed view-construction rule with a learned instance-wise augmentation module. The designer does not predict segmentation masks or depth maps. Instead, it modifies the self-supervised views used by DINO so that the resulting SSL gradient better aligns with dense-task feedback gradients.

Augmentor parameterization. The task designer is an image augmentor a_ϕ that predicts a soft spatial mask $\mu_\phi(v) \in [0,1]^{H \times W}$ for selected SSL crops. Two augmentor architectures are supported: a small convolutional U-Net [65] mask generator and a small dit-based [53] patchwise mask generator. In the default configuration, the augmentor is applied only to the two global SSL crops, while the local crops remain generated by the standard DINO pipeline.

Given a base crop v, the learned view is formed by softly mixing the original crop with a blurred version:

$$\tilde{v} = \mu_\phi(v) \odot v + (1 - \mu_\phi(v)) \odot \mathrm{Blur}(v). \tag{49}$$

Large mask values preserve the original pixels; small values suppress pixels by replacing them with a blurred background. For an image x, let $V_\phi(x)$ denote the resulting set of views after replacing the controlled global crops with learned views. The SSL loss under the designer is

$$L_{\mathrm{ssl}}(\theta;\phi,B^u)=\frac{1}{|B^u|}\sum_{x\in B^u}\ell_{\mathrm{DINO}}\big(\theta,\bar{\theta};V_\phi(x)\big)\,. \tag{50}$$

Dense feedback tasks. The feedback signal comes from small labeled subsets of ADE20K and NYUv2. For the default feedback construction, we use ADE20K / SceneParse150 with 2,000 labeled training samples and 512 meta samples, and NYUv2 with 512 labeled training samples and 128 meta samples. The labeled training samples are used to fit lightweight downstream evaluator heads, and the meta samples are used to compute downstream gradients for the task designer.

The evaluator heads are:

$$H_{\mathrm{seg}}\quad\text{for ADE20K semantic segmentation,}\qquad H_{\mathrm{dep}}\quad\text{for NYUv2 depth estimation.}$$

The segmentation head is trained with pixelwise cross-entropy. The depth head is trained with an L1 loss on valid depth pixels.

Meta step. A V-pretraining meta step (task designer update step) has three stages.

First, the downstream evaluator heads are updated on small labeled training batches using frozen backbone features. This adapts the lightweight heads to the current representation without applying supervised dense-task updates to the backbone.

Second, on separate labeled meta batches, we compute a downstream evaluator loss

$$L^{\mathrm{vis}}_{\mathrm{down}}(\theta;B^f)=\alpha_{\mathrm{seg}}L_{\mathrm{seg}}\big(H_{\mathrm{seg}}(F_\theta(x)),y_{\mathrm{seg}}\big)+\alpha_{\mathrm{dep}}L_{\mathrm{dep}}\big(H_{\mathrm{dep}}(F_\theta(x)),y_{\mathrm{dep}}\big), \tag{51}$$

with terms omitted when the corresponding feedback task is not used. Here F_θ is the visual backbone, and $\alpha_{\mathrm{seg}},\alpha_{\mathrm{dep}}$ control the relative value placed on segmentation and depth. We compute the downstream gradient with respect to a selected subset S of backbone parameters:

$$g^{\mathrm{vis}}_{\mathrm{down,S}}=\mathrm{sg}\big(\nabla_{\theta_S}L^{\mathrm{vis}}_{\mathrm{down}}(\theta;B^f)\big)\,. \tag{52}$$

In practice, S is chosen as the last K transformer blocks together with final normalization layers. This reduces the cost of the mixed second-order gradient while keeping the alignment signal tied to the representation layers used by dense prediction.

Third, a separate unlabeled ImageNet batch is passed through the learned augmentor, the DINO loss is recomputed under the learned views, and the SSL gradient is computed on the same parameter subset:

$$g^{\mathrm{vis}}_{\mathrm{ssl,S}}=\nabla_{\theta_S}L_{\mathrm{ssl}}(\theta;\phi,B^u). \tag{53}$$

The augmentor is trained to maximize alignment between the downstream dense-task gradient and the SSL gradient:

$$L^{\mathrm{vis}}_{\mathrm{meta}}(\phi)=-\left\langle g^{\mathrm{vis}}_{\mathrm{down,S}},g^{\mathrm{vis}}_{\mathrm{ssl,S}}\right\rangle+\lambda_{\mathrm{spars}}R_{\mathrm{spars}}(\mu_\phi)+\lambda_{\mathrm{tv}}R_{\mathrm{tv}}(\mu_\phi). \tag{54}$$

The sparsity regularizer keeps the mean mask value near a target keep ratio, and the total-variation regularizer encourages spatially smooth masks. These regularizers keep the learned augmentation close to plausible SSL view construction rather than arbitrary image corruption.

Learner update. Only the augmentor is updated by $L^{\mathrm{vis}}_{\mathrm{meta}}$. After the augmentor update, the views are recomputed and detached. The backbone learner is then updated by an ordinary DINO SSL loss:

$$g^{\mathrm{learn}}_t=\nabla_\theta L_{\mathrm{ssl}}\big(\theta_t;\mathrm{sg}(V_{\phi_t}(B^u_t))\big),\qquad \theta_{t+1}=\theta_t-\eta_t g^{\mathrm{learn}}_t. \tag{55}$$

Thus ADE20K and NYUv2 labels never appear as supervised learner targets during continued pretraining. They train only the task designer that constructs SSL views.

Second-order implementation detail. The meta update differentiates through the SSL gradient with respect to selected backbone parameters. This requires second-order differentiation through the backbone. To avoid unsupported double-backward paths in Flash or memory-efficient attention kernels, the implementation uses a math-only scaled dot-product attention backend during the relevant meta-gradient computation.

G.3 Dense evaluation protocol

ADE20K semantic segmentation. ADE20K evaluation uses the SceneParse150 dataset. Label 0 is remapped to the ignore index 255, and labels $1, \ldots, 150$ are remapped to $0, \ldots, 149$. Performance is reported as mean intersection-over-union (mIoU).

The evaluation code supports both a small convolutional segmentation decoder and a linear-BN segmentation head. In the sweep comparisons used for the main reported results, the evaluation protocol is fixed to a frozen-backbone linear probe with a linear-BN head. This makes the comparison primarily a test of representation quality rather than downstream fine-tuning sensitivity.

NYUv2 depth estimation. NYUv2 evaluation uses the standard validation split. Depth values are decoded to meters, and metrics are computed on valid pixels. The evaluation protocol enables the standard Eigen crop by default. The main metric reported in the paper is RMSE; the evaluation code also computes AbsRel and δ_1.

As with segmentation, the sweep comparisons use a frozen-backbone linear probing protocol with a linear-BN depth head. Predictions are passed through a softplus nonlinearity to ensure positive depth values.

ImageNet linear evaluation and transfer. We additionally evaluate whether dense-task feedback harms global recognition. For this purpose, the main table reports ImageNet-1K linear accuracy. We also report instance-retrieval transfer on Revisited Oxford and Revisited Paris. These tasks are not used as feedback during V-pretraining and are used to check whether dense feedback collapses the representation onto segmentation or depth.

Evaluation hyperparameters. All selected checkpoints are evaluated at the last pretraining checkpoint (`eval_pretrain_steps=last`) with a frozen-backbone linear probing protocol (`eval_ft_mode=linear`) and a linear-BN head. Evaluation uses 20k optimization steps, batch size 8, bf16 precision, 1k warmup steps, weight decay 0.05, and gradient clipping at 1.0. ADE20K evaluation uses image size 512 and head learning rate 5×10^{-4}. NYUv2 depth evaluation uses an L2 depth loss, head learning rate 10^{-3}, resize resolution 480×640, valid depth range $[10^{-3}, 10]$ meters, and the standard Eigen crop.

G.4 Hyperparameter selection

Fixed-budget sweep protocol. Vision hyperparameters are selected with fixed-budget pretraining sweeps. Each sweep trial performs 20k steps of continued DINO pretraining and then evaluates one or more saved checkpoints, with the default configuration evaluating the last checkpoint. ADE20K-oriented sweeps optimize validation mIoU, while NYUv2-oriented sweeps optimize validation RMSE.

The main swept hyperparameters are:

learner learning rate	η,
meta-update frequency	r,
meta SSL batch size	$\lvert B^u_{\mathrm{meta}}\rvert$,
aligned backbone blocks	K,
augmentor architecture	U-Net or tiny DiT,
augmentor learning rate	η_ϕ,
target mask keep ratio	κ,
sparsity regularization	λ_{spars},
total-variation regularization	λ_{tv},
segmentation feedback weight	α_{seg},
depth feedback weight	α_{dep}.

Backbone-scale adjustments. The sweep spaces are adjusted for backbone size to fit memory. For ViT-B, the pretraining batch size is larger; for ViT-L, the pretraining batch size and meta-SSL batch-size search space are reduced. In the larger depth sweep, the augmentor architecture is fixed to the U-Net mask generator to reduce search cost. These adjustments affect the feasible training configuration, not the definition of the feedback channel.

Target	Backbone	LR	Batch	Meta freq.	Meta SSL batch	Aligned blocks	Augmentor	Aug. LR	Keep	λ_{spars}	λ_{tv}	α_{seg}	α_{dep}
ADE20K	ViT-B	4.15×10^{-6}	256	4	32	2	U-Net	1.54×10^{-4}	0.6	0.131	0.144	1	8
ADE20K	ViT-L	4.50×10^{-5}	50	4	16	4	U-Net	2.17×10^{-4}	0.5	0.196	0.0399	4	1
NYUv2	ViT-B	8.58×10^{-6}	256	8	64	3	SiT	2.21×10^{-4}	0.4	0.111	0.0417	16	2
NYUv2	ViT-L	1.94×10^{-6}	50	2	32	4	U-Net	8.03×10^{-4}	0.4	0.401	0.159	32	16

Table 11: Selected V-pretraining hyperparameters for the main DINOv3 vision runs. Batch is the learner SSL batch size. Meta freq. is the number of learner steps between augmentor updates. Meta SSL batch is the unlabeled ImageNet batch used to compute the SSL gradient in the value objective. Aligned blocks denotes the number of final ViT blocks used for gradient alignment. The target column denotes the sweep selection metric, not the only feedback task used by the value objective.

Interpretation of vision tuning. The vision results should be interpreted under this stated tuning protocol: standard continued DINO pretraining and V-pretraining are compared with the same initial backbone, unlabeled ImageNet stream, SSL objective family, and evaluation protocol, while V-pretraining additionally pays the cost of learned view construction and periodic value-gradient updates. The purpose of the experiment is not to introduce a new dense-prediction fine-tuning method, but to test whether dense feedback can improve the self-supervised pretraining updates that shape the backbone representation.

Selected V-pretraining configurations. Table 11 reports the selected V-pretraining configurations used for the main DINOv3 vision results. The target column denotes the validation metric used to select the run from the fixed-budget sweep; it does not mean that only that feedback task is present in the meta objective. When both dense feedback loaders are enabled, the downstream evaluator gradient is formed from the weighted segmentation and depth losses with weights α_{seg} and α_{dep}.

All selected runs use 20k continued-pretraining steps, bf16 mixed precision, AdamW, weight decay 0.04, gradient clipping at 1.0, 1k warmup steps, DINO output dimension 8192, student temperature 0.1, teacher temperature 0.04, teacher momentum base 0.996, center momentum 0.9, two global crops of size 224, six local crops of size 96, and learned augmentation applied only to the global crops. The unlabeled stream is ImageNet-1K, and the feedback pools use 2,000 ADE20K training samples with 512 ADE20K meta samples and 512 NYUv2 training samples with 128 NYUv2 meta samples.

H Declarations and Impacts

Use of generative AI tools. We used ChatGPT Image to draft the illustrative schematic in Figure 1 from author-written prompts. The figure contains no experimental data, generated results, or method output. We also used ChatGPT to polish some parts of the writing. All equations, labels, and scientific content were manually checked by the authors. Our use of AI tools follow the NeurIPS guidelines.

Broader Impacts. As V-pretraining fundamentally enhances the capabilities of machine learning models, it inherently carries a dual-use nature. On the positive side, our approach can significantly benefit downstream applications such as creative content generation and advanced information retrieval. Conversely, we acknowledge the potential risks of malicious exploitation, particularly regarding the generation of deceptive, fake, or misleading content.

I Compute Resources and Total Compute

Compute workers. Training experiments were run on internal GPU clusters and on the DeltaAI system at the National Center for Supercomputing Applications, whose nodes use NVIDIA GH200 Grace Hopper superchips with H100 GPUs. On the internal clusters, the main training workers were NVIDIA H100 80GB and NVIDIA H200 NVL GPUs. Auxiliary evaluation, small diagnostic runs, and some post-training comparisons also used RTX 6000 Ada / RTX A6000-class GPUs with 48GB memory. We report compute in accelerator-hours, counting one hour on one GPU as one accelerator-hour. Most reported runs were single-worker training jobs; when a node exposed multiple GPUs, we count only the GPUs used by the corresponding training process.

Experiment family	Main worker	Approx. compute per run	Memory used	Notes
Qwen 0.5B language continued pretraining	1 H100/H200	2–6 GPU-hours	30–40GB	Baseline or V-pretraining run; GSM8K feedback for V-pretraining.
Qwen 4B language continued pretraining	1–2 H100/H200	10–24 GPU-hours	60–140GB	Used for main GSM8K result, ablations, and decontamination controls.
Qwen 7B language continued pretraining	1–2 H100/H200	16–36 GPU-hours	80–140GB	Used for scaling runs.
Language multitask run	1 H100/H200	4–12 GPU-hours	30–50GB	Qwen2.5-0.5B multitask pretraining with math/code/general feedback.
DINOv3 ViT-B continued SSL	1 H100/H200	1–3 GPU-days	50–80GB	20k-step continued pretraining plus dense linear-probe evaluation.
DINOv3 ViT-L continued SSL	1 H100/H200	1–4 GPU-days	80–140GB	20k-step continued pretraining plus dense linear-probe evaluation.
Dense evaluation / retrieval / linear probes	H100/H200 or RTX 6000 Ada	0.5–6 GPU-hours	20–48GB	ADE20K, NYUv2, ImageNet linear, and retrieval evaluations.
Direct-feedback diagnostic	H100 and RTX 6000 Ada	10 min–16 GPU-hours	20–80GB	Small V-pretraining/baseline diagnostic and SFT+GRPO comparison.

Table 12: Approximate compute for individual reported experimental runs. GPU-hours denote accelerator-hours. Ranges reflect differences in model size, device type, and whether evaluation is included in the same job. The reported main comparisons are wall-clock matched within each experiment family, so V-pretraining is charged for its task-designer and value-gradient overhead.

Training jobs used CPU workers for data loading. Vision pretraining jobs used 12 dataloader workers in the selected configurations. Slurm allocations typically provided 12–16 CPU cores per GPU worker and 80–142GB of allocated host memory, with larger nodes providing substantially more physical RAM. Datasets and checkpoints were stored on shared cluster storage; individual runs wrote checkpoints, logs, and evaluation outputs, typically requiring tens of GB per run. The full project required on the order of 0.5–1TB of shared storage across datasets, checkpoints, logs, sweeps, and intermediate evaluation outputs.

Per-run compute. Table 12 summarizes the approximate compute for the reported experimental runs. The numbers are estimates from job logs and observed throughput. They are intended to document the scale of the experiments rather than serve as exact hardware benchmarks, since wall-clock time varies with device type, dataloader throughput, cluster load, and whether evaluation is run in the same job.

Representative measured overhead. For the current language V-pretraining configuration, the task designer is a small 256-hidden-dimensional transformer with 6 layers and 8 heads, aligning the last two learner layers and performing a meta update every 8 learner optimizer steps. In a representative current run, the standard baseline used microbatch size 4, gradient accumulation 32, and sequence length 512, for $4 \times 32 \times 512 = 65{,}536$ tokens per optimizer step. The corresponding V-pretraining run used microbatch size 4, gradient accumulation 20, and sequence length 512, for $4 \times 20 \times 512 = 40{,}960$ tokens per optimizer step. The baseline achieved approximately 26.7k tokens/s with peak GPU memory 31,302 MiB, while V-pretraining achieved approximately 24.2k tokens/s with peak GPU memory 36,440 MiB. Thus the current V-pretraining implementation reduces token throughput by about 9% and increases peak memory by about 16% in this representative language setting. The shorter V-pretraining optimizer-step time is not evidence of lower cost, since each optimizer step processes fewer tokens.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims are supported by experimental results in Section 3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: See Appendix B and Appendix B.3.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The hyperparameters are reported in Appendix F and Appendix G. We provide code in the supplementary files.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code in the supplementary files.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Please see Appendix F, Appendix G, and the attached code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A]

   Justification: For some experiments, we provide error bars. However, V-pretraining incurs a high computational cost, which limits the feasibility of running multiple trials for every experiment within a reasonable timeframe.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix I.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We followed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Appendix H.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: We will do responsible AI reviews before releasing the models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

    Justification: All the code and models of this work is created by us.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The code comes with a README.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We do not have such experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not have such experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: See Appendix H.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.