Olmo 3

Olmo Team[*]

Allyson Ettinger[♥,1]   Amanda Bertsch[♥,1,3]   Bailey Kuehl[♥,1]   David Graham[♥,1]
David Heineman[♥,1]   Dirk Groeneveld[♥,1]   Faeze Brahman[♥,1]   Finbarr Timbers[♥,1]
Hamish Ivison[♥,1,2]   Jacob Morrison[♥,1,2]   Jake Poznanski[♥,1]   Kyle Lo[♥,1,2]   Luca Soldaini[♥,1]
Matt Jordan[♥,1]   Mayee Chen[♥,1,4]   Michael Noukhovitch[♥,1,5,6]   Nathan Lambert[♥,1]
Pete Walsh[♥,1]   Pradeep Dasigi[♥,1]   Robert Berry[♥,1]   Saumya Malik[♥,1]   Saurabh Shah[♥,1]
Scott Geng[♥,1,2]   Shane Arora[♥,1]   Shashank Gupta[♥,1]   Taira Anderson[♥,1]   Teng Xiao[♥,1]
Tyler Murray[♥,1]   Tyler Romero[♥,1]   Victoria Graf[♥,1,2]

Akari Asai[1,3]   Akshita Bhagia[1]   Alex Wettig[7]   Alisa Liu[2]   Aman Rangapur[1]
Chloe Anastasiades[1]   Costa Huang[1]   Dustin Schwenk[1]   Harsh Trivedi[1]   Ian Magnusson[1,2]
Jaron Lochner[1]   Jiacheng Liu[1]   Lj Miranda[1]   Maarten Sap[1,3]   Malia Morgan[1]
Michael Schmitz[1]   Michal Guerquin[1]   Michael Wilson[1]   Regan Huff[1]   Ronan Le Bras[1]
Rui Xin[2]   Rulin Shao[2]   Sam Skjonsberg[1]   Shannon Zejiang Shen[8]   Shuyue Stella Li[2]
Tucker Wilde[1]   Valentina Pyatkin[1]   Will Merrill[1]   Yapei Chang[9]   Yuling Gu[1]   Zhiyuan Zeng[1,2]

Ashish Sabharwal[1]   Luke Zettlemoyer[2]   Pang Wei Koh[1,2]
Ali Farhadi[1,2]   Noah A. Smith[♥,1,2]   Hannaneh Hajishirzi[♥,1,2]

[1]Allen Institute for AI   [2]University of Washington   [3]Carnegie Mellon University   [4]Stanford University   [5]Mila
[6]Université de Montréal   [7]Princeton University   [8]Massachusetts Institute of Technology   [9]University of Maryland

[*]OLMO 3 was a team effort; authors sorted alphabetically. ♥marks core contributors.

🙋 **Olmo 3 Base:** `Olmo-3-1025-7B` `Olmo-3-1125-32B`
🙋 **Olmo 3 Think:** `Olmo-3-7B-Think` `Olmo-3-32B-Think`
🙋 **Olmo 3 Instruct:** `Olmo-3-7B-Instruct`
🙋 **Olmo 3 RL Zero:** `Olmo-3-7B-RLZero-Math, Olmo-3-7B-RLZero-Code, Olmo-3-7B-RLZero-IF, Olmo-3-7B-RLZero-Mix`

🗄 **Base Data:** `Pretrain: Dolma 3 Mix`   `Midtrain: Dolma 3 Dolmino Mix`   `Long-ctx: Dolma 3 Longmino Mix`

🗄 **Think Data:** `SFT: Dolci-Think-SFT`   `DPO: Dolci-Think-DPO`   `RL: Dolci-Think-RL`

🗄 **Instruct Data:** `SFT: Dolci-Instruct-SFT`   `DPO: Dolci-Instruct-DPO`   `RL: Dolci-Instruct-RL-7B`

🗄 **RL Zero Data:** `Dolci-RLZero-Math-7B`   `Dolci-RLZero-Code-7B`   `Dolci-RLZero-IF-7B`   `Dolci-RLZero-Mix-7B`

🐙 **Training Code:** `OLMo-core` (pretrain)   `Open Instruct` (posttrain)
🐙 **Data Code:** `datamap-rs` (data processing)   `duplodocus` (deduplication)   `dolma3` (data recipes)
🐙 **Eval Code:** `OLMES` (eval suite)

✿ **Demo:** 32B Think   7B Think   7B Instruct

✉ **Contact:** `olmo@allenai.org`

Abstract   ✿Ai2

We introduce **Olmo 3**, a family of state-of-the-art, fully open language models at the 7B and 32B parameter scales. OLMO 3 model construction targets long context reasoning, function calling, coding, instruction following, general chat, and knowledge recall. The release includes the entire **model flow**, i.e., the full lifecycle of the family of models, including every stage, checkpoint, datapoint, and dependency used to build it. Our flagship model, **Olmo 3 Think-32B**, is the strongest fully open thinking model released to-date.
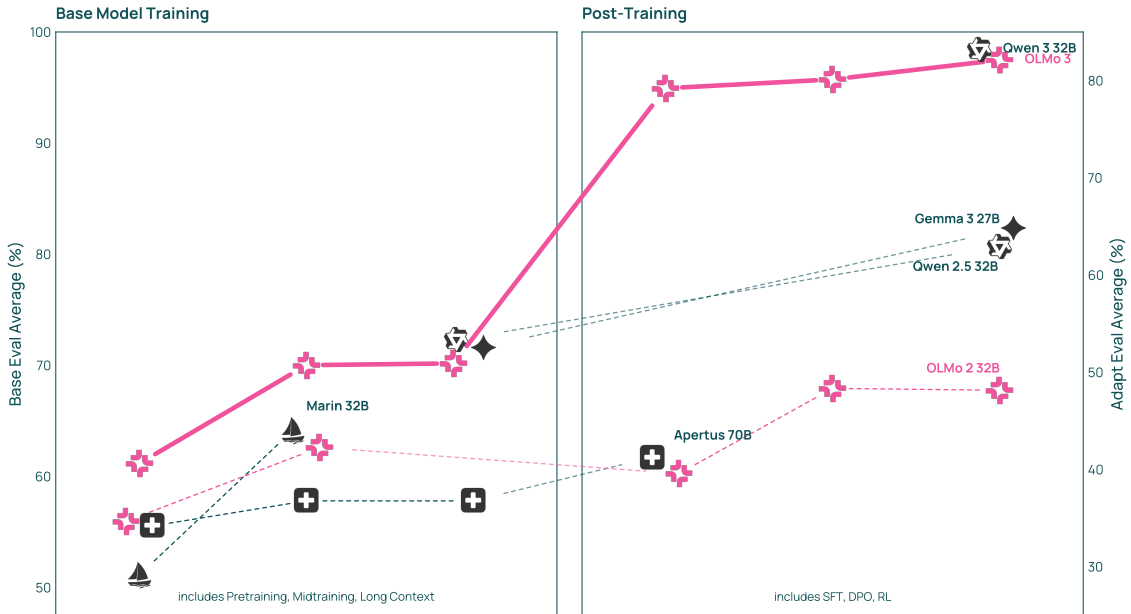
1

Contents

1 Introduction

We introduce **Olmo 3**, a family of state-of-the-art, fully open language and thinking models at the 7B and 32B parameter scales with a diverse set of capabilities, including long context reasoning, function calling, coding, instruction following, general chat, and knowledge recall. With the OLMO 3 release, we go farther by providing complete access to its entire **model flow**: the full life-cycle of a language model, including every stage, checkpoint, datapoint, and dependency required to create it. This enables infinite customization through intervention at any stage of the model development process—not just the final weights.
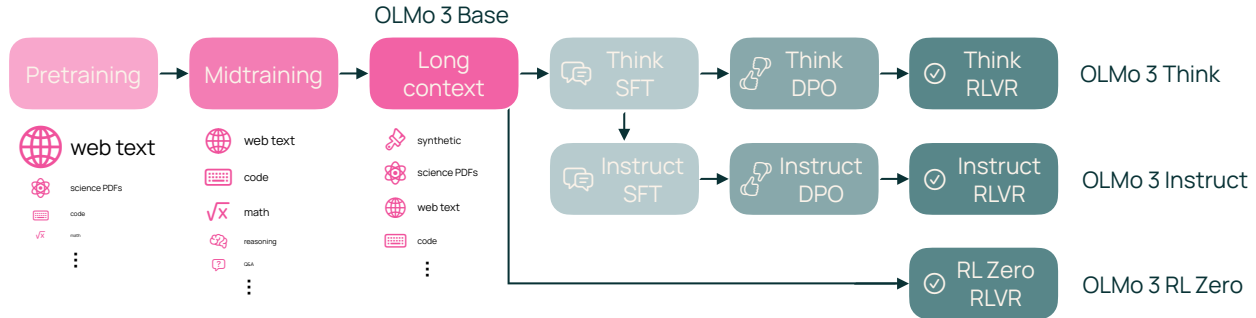
To truly advance open-source AI research and development, we argue that releasing a state-of-the-art language model should make its entire model flow—not just its endpoint—transparent and accessible. With the OLMO 3 release, we provide complete access to the pathways we charted throughout the model flow, from initial conception to the creation of state-of-the-art, fully open language models.

Specifically, we train **Olmo 3 Base** as a foundation on which to build models for thinking and tool use capabilities. From OLMO 3 BASE we develop our flagship model, **Olmo 3 Think**, trained to perform step-by-step reasoning by generating intermediate thinking traces before producing a final answer. OLMO 3 THINK-32B is the strongest fully open thinking model, narrowing the gap to the best open-weight models of similar scale, such as the Qwen 3-32B thinking models (Yang et al., 2025a), on our suite of reasoning benchmarks, while being trained on six times fewer tokens. For the first time, the OLMO 3 release also enables reasoning chains to be traced back to their original training data.

Figure 1 In OLMO 3, we release the full model flow, including data, code and weights for all stages of development. OLMO 3 THINK 32B (shown here) is competitive with Qwen 3 32B which does not have a released base model. Our underlying OLMO 3 BASE 32B is comparable with the best open-weights base models like Qwen 2.5 32B and Gemma 3 27B, but it additionally comes with intermediate checkpoints that enable the study of mid-training and long context extension recipes.

In addition, we train **Olmo 3 Instruct**, which produces shorter sequences than the corresponding OLMO 3 THINK model to improve inference efficiency and is designed to focus on general chat and function calling. **Olmo 3 Instruct** outperforms open weight models Qwen 2.5 (Qwen et al., 2024), Gemma 3 (Gemma 3 Team, 2025), Llama 3 (Grattafiori et al., 2024), and narrows the gap to Qwen 3 (Yang et al., 2025a) model families at a similar scale. The Olmo 3 family is the best collection of fully open base models, outperforming Stanford Marin (Hall et al., 2025), Apertus (Apertus Team, 2025), and LLM360 (Liu et al., 2025c). Finally, we introduce **Olmo 3 RL-Zero** with RL training on top of our base model to enable researchers with RL benchmarking with open pretraining data.

Figure 2 Depiction of OLMO 3's model flow. Development is divided into major pretraining and post-training stages, each further divided into sub-stages with their own recipes (i.e., training data and method).

The Olmo 3 family is the strongest collection of fully open base models, outperforming Stanford Marin (Hall et al., 2025), Apertus (Apertus Team, 2025), and LLM360 (Liu et al., 2025c). To achieve these results, we construct new cutting-edge datasets for every stage of the model training pipeline. This includes **Dolma 3**, our fine-grained pretraining data mix encompassing heavily deduplicated natural data from the web, our midtraining mix of high-quality data designed to jump-start reasoning, and our new long-context training mix. We also introduce **Dolci**, a post-training data suite that advances step-by-step reasoning during supervised finetuning, provides high-quality contrastive data for preference tuning, and offers challenging general and reasoning prompts for reinforcement learning.

In addition, we develop a set of new algorithmic and infrastructure advances across data processing, evaluation, pretraining, and reinforcement learning. This includes **OlmoBaseEval**, a suite of benchmarks specifically for making base model development decisions at compute-efficient scales, and **OlmoRL**, our RL training approach that features algorithmic and infrastructural improvements targeted in particular at training our thinking models. To design these training recipes and coordinate targeted improvements across a wide range of capabilities at each stage of the model training pipeline, our development framework balances distributed innovation with centralized evaluation.

2 Model Flow for Olmo 3

In this section, we provide a brief overview of all of the components of the OLMO 3 model flow, highlighting our methodology in targeting reasoning and tool use capabilities in ways that advance beyond OLMO 2 (OLMO et al., 2024) and other open weight models. Subsequent sections will then provide deep dives on each of the model flow components. OLMO 3 training is divided into major stages of pretraining and post-training, each of which is further divided into sub-stages as outlined in Figure 2.

2.1 Base Model Training

We develop OLMO 3 BASE in three stages of *pretraining* for up to 5.9T tokens (Section 3), *mid-training* for 100 billion tokens (Section 3.5.1), and the newly-added *long-context extension* for 50 (OLMO 3 7B) or 100 (OLMO 3 32B) billion tokens (Section 3.6).

Evaluation We develop **OLMOBASEEVAL**, a collection of benchmarking suites to support decision-making during base model development (pretraining and midtraining). Our goal is to be compute-efficient by making development decisions based on models trained at a small scale. The challenge is that such models can exhibit random-chance performance on certain tasks, with small differences in scores that are hard to distinguish from benchmark noise. To address this, we (1) aggregate scores over clusters of tasks that assess similar capabilities (§3.3.1); (2) develop proxy metrics for evaluating small-scale models (§3.3.2); and (3) improve overall signal-to-noise ratio by evaluating on more examples from noisy tasks or even removing them entirely (§3.3.3).

Data Curriculum   We curate specialized datasets for each training stage, with increasing focus on core capabilities that enable enabling downstream elicitation of target capabilities, like thinking, instruction-following, and others:

- **Pretraining:** We first train Olmo 3 Base on **Dolma 3 Mix**, our 6T token pretraining data mix. While Dolma 3 Mix is comprised of largely the same types of data sources used in other open pretraining recipes (Soldaini et al., 2024; Bakouch et al., 2025; OLMo et al., 2024), we demonstrate three key novelties:
  - New tooling for fast and scalable global deduplication at the trillion-token scale,
  - Two new methods for optimizing selection of training tokens: token-constrained mixing and quality-aware upsampling,
  - A novel source of academic PDFs—olmOCR Science PDFs—converted to linearized plain text using olmOCR (Poznanski et al., 2025a) (§3.4.2),

- **Midtraining:** We continue training on **Dolma 3 Dolmino Mix** (§3.5), our 100B data curated to boost target capabilities across code, math, and general knowledge QA domains through the introduction of:
  - A new two-part methodological framework combining 1) lightweight, distributed feedback loops on individual data sources, with 2) centralized integration tests to assess candidate mixes on base model quality and post-trainability.
  - More intentional inclusion of data types—instruction data and thinking traces—to lay groundwork for supporting post-trainining.

- **Long-context extension (Dolma 3 Longmino Mix):** We curate high-quality data, including a large collection of scientific PDFs processed with our state-of-the-art OCR tool, olmOCR (Poznanski et al., 2025a,b). These PDFs are incorporated throughout all stages of base model training but form an especially important component of our long-context extension training data; with over 4.5 million documents with length above 32K totaling over 380 billion tokens, this collection is the largest openly available for long-context researchers. Olmo 3 is our first model with long-context capabilities, supporting up to 64K context after extension.

Open Artifacts   We release all our intermediate checkpoints as well as the final models at the end of each stage of training. We release not just our training data, but also release both **data pools**, 10T tokens of cleaned source tokens, and **data mixes**, which are the actual tokens used for base model training.[1] For data mixes, we release both our actual training mix for Olmo 3 Base as well as a smaller sample for accessible experimentation with less compute (e.g., both 5.9T mix and 100B mixes for Dolma 3 Mix).

2.2   Post-training

Figure 2 shows our post-training pipeline for the Olmo 3 model flow tracking three pathways. This pipeline support the training of Olmo 3 Think (§4), and Olmo 3 Instruct (§5), each of which consists of a three-stage training pathway, as well as training of Olmo 3 RL-Zero (§6), which applies reinforcement learning training directly following the base model training. For post-training, we introduce Dolci, a new state-of-the-art dataset with tailored data at each stage of post-training to address thinking and function calling, and Olmo 3-RL, our new algorithmic advances to RLVR with long thinking traces.

- **Olmo 3 Think** (§4) is trained for reasoning by generating extended thinking before producing a final answer. To achieve this, we employ a three-stage post-training process comprising Supervised Fine-Tuning (SFT), Preference Fine-Tuning via Direct Preference Optimization (DPO), and then Reinforcement Learning with Verifiable Rewards (RLVR). We observe consistent gains across all three stages, demonstrating the impact of careful data curation, algorithmic refinement, and infrastructure engineering.
  - We introduce Dolci Think SFT (§4.2), Dolci Think DPO (§4.3), and Dolci Think RL (§4.4), new cutting-edge post-training datasets designed to target a broad range of key capabilities such as math, coding, instruction following, and general conversation. The dataset includes synthetic examples with long thinking traces for supervised fine-tuning, high-quality contrastive data following the insights from Delta Learning Geng et al. (2025), and challenging prompts for reinforcement learning across both

---

[1]A data mix may involve upsampling or repeating data from a data pool.

verifiable and non-verifiable domains. Notably, our new approach to curate contrastive instances for preference tuning expands the reasoning frontier of the model beyond what SFT alone can provide and primes the model for effective reinforcement learning.

– We introduce algorithmic and infrastructure advances in reinforcement learning with verifiable rewards (§4.4). This approach generalizes verifiable reasoning to multiple domains, expanding beyond the settings explored in OLMo 2 to include code and general chat. Our improvements enable longer and more stable RL runs across diverse domains and increase the overall efficiency of training cycles, leading to a 4x speedup in RL training.

- **Olmo 3 Instruct (§5)** is trained to produce efficient and helpful responses to user queries without generating internal thinking traces. This model prioritizes typical user needs, such as avoiding excessive verbosity for easy user understanding and function-calling for user information seeking. In such settings, thinking traces are unnecessary and inference-time efficiency matters more than inference-time scaling.

  We introduce Dolci Instruct SFT, our new dataset enriched with data specifically created for function calling (Section §5.2.1). To directly optimize model interactivity on top of capabilities, we extend our Delta Learning preference pipeline in Dolci Instruct DPO, incorporating multi-turn preference data and targeted data length interventions that encourage concise responses (Section §5.3.1). Finally, we use reinforcement learning with verifiable rewards (Section §5.4) to further refine core capabilities, where preference tuning synergizes with RL to improve model performance while maintaining learned brevity.

- **Olmo 3 RL-Zero (§6)** To date, all leading open RLVR benchmarks and algorithms train on top of open-weights models that do not reveal their pretraining or mid-training data (Chu et al., 2025; Yang et al., 2025a). This limits the ability for the community to study the role of pretraining data on RLVR performance. It can lead to a myriad of issues with benchmark evaluations being contaminated e.g. mid-training data containing the evaluation which makes spurious rewards as effective as true reward (Shao et al., 2025; Wu et al., 2025c) or improvements from fixing prompt templates outweighing the improvements from RL (Liu et al., 2025b).

  We therefore release a fully open dataset Dolci RL-Zero, an algorithmic RL zero setup for Olmo 3, and open-source OlmoRL code to enable clear benchmarking for the ecosystem. We perform RLVR from Olmo 3 Base over four benchmarking domains to create the Olmo 3 RL-Zero family: math, code, precise instruction following (IF) and a general mix. In all cases, we further decontaminate Dolci RL-Zero from pretraining and midtraining data to guarantee our setup carefully studies the effect of RLVR without data leakage confounding our conclusions.

2.3 Results

Table 13 demonstrates a snapshot of our evaluation for Olmo 3 Think compared to other open-weight and fully open models. Refer to Section §3.1 for evaluation of Olmo 3 Base, Section §5.1 for Olmo 3 Instruct, and Section §4.1 for details about Olmo 3 Think.

Olmo 3 Think is the strongest fully open thinking model we are aware of. It is better than Qwen2.5-Instruct, Gemma 2 and 3 27B, DeepSeek R1 Distilled Qwen 32B; it is also close to Qwen 3 and Qwen 3 VL 32B models. narrowing the gap to the best open-weight models of similar scale – such as Qwen 3 32B – while training on roughly 6x fewer tokens.

3 Olmo 3 Base

The goal of Olmo 3 Base is to establish a strong foundation that supports a diversity of general capabilities while enabling downstream capabilities like thinking, tool-use, and instruction-following to be easily elicited during post-training. In this section, we describe our recipe for Olmo 3 Base, organized as follows:

- **Modeling (§3.2)** Olmo 3 Base closely follows OLMo 2 in that it is a dense model at 7B and 32B sizes, with largely identical hyperparameters. Apart from engineering improvements that enable better throughput, we focus on enabling a larger context window. We lay out the details in §3.2.
- **Evaluation (§3.3)** To guard against overfitting the base model to any one capability, we greatly expand on

Benchmark	Olmo 3 Think 32B SFT	Olmo 3 Think 32B DPO	Olmo 3 Think 32B	Qwen 3 32B	Qwen 3 VL 32B Thinking	Qwen 2.5 32B	Gemma 3 27B Instruct	Gemma 2 27B Instruct	Olmo 2 32B Instruct	DeepSeek-R1-Distill-Qwen-32B
Math										
MATH	95.6	95.9	96.1	95.4	96.7	80.2	87.4	51.5	49.2	92.6
AIME 2024	73.5	76.0	76.8	80.8	86.3	15.7	28.9	4.7	4.6	70.3
AIME 2025	66.2	70.7	72.5	70.9	78.8	13.4	22.9	0.9	0.9	56.3
OMEGA	43.1	45.2	50.8	47.7	50.8	19.2	24.0	9.1	9.8	38.9
Reasoning										
BigBenchHard	88.8	89.1	89.8	90.6	91.1	80.9	82.4	66.0	65.6	89.7
ZebraLogic	70.5	74.5	76.0	88.3	96.1	24.1	24.8	17.2	13.3	69.4
AGI Eval English	85.9	87.8	88.2	90.0	92.2	78.9	76.9	70.9	68.4	88.1
Coding										
HumanEvalPlus	90.0	91.6	91.4	91.2	90.6	82.6	79.2	67.5	44.4	92.3
MBPP+	66.7	67.2	68.0	70.6	66.2	66.6	65.7	61.2	49.0	70.1
LiveCodeBench v3	75.8	81.9	83.5	90.2	84.8	49.9	39.0	28.7	10.6	79.5
IF										
IFEval	83.9	80.6	89.0	86.5	85.5	81.9	85.4	62.1	85.8	78.7
IFBench	37.0	34.4	47.6	37.3	55.1	36.7	31.3	27.8	36.4	23.8
Knowledge & QA										
MMLU	85.3	85.2	85.4	88.8	90.1	84.6	74.6	76.1	77.1	88.0
PopQA	33.1	37.0	31.9	30.7	32.2	28.0	30.2	30.4	37.2	26.7
GPQA	55.7	57.6	58.1	67.3	67.4	44.6	45.0	39.9	36.4	61.8
Chat										
AlpacaEval 2 LC	69.1	78.6	74.2	75.6	80.9	81.9	65.5	39.8	38.0	26.2
Safety	64.8	65.3	68.8	69.0	82.7	81.9	68.6	74.3	83.8	63.6

Table 1 Results on our flagship model Olmo 3 Think-32B on our Olmo 3 Eval suite. Olmo 3 Think-32B is the best fully open model at 32B.

our evaluation suite from OLMo 2 to include more benchmarks. We also improve our ability to rely on small scale experiments by improving how we select and use benchmarks throughout development. We discuss this more in §3.3.

- **Data** We introduce **Dolma 3**, a collection of data to support multiple stages of base model development:

  1. **Pretraining (§3.4)** We train on **Dolma 3 Mix**, 5.9T tokens of diverse, natural data including sources like web pages, academic PDFs, code repositories, and more. We trained on 1024 H100s.[2] Our training throughput achieves 7.7K tokens / device / second for Olmo 3 Base-7B.

  2. **Midtraining (§3.5)** We train on **Dolma 3 Dolmino Mix**, 100B tokens combining our highest-quality pretraining data with substantial task data for math and code problems, instruction following, reading comprehension, and more. We trained on 128 H100 GPUs.[3]

  3. **Long-context extension (§3.6)** We train on **Dolma 3 Longmino Mix**, 50B / 100B tokens combining long documents with our midtraining data. We trained on 256 H100s.[4] Long-context extension required special training considerations, which we detail in §3.6.

3.1 Main Results for Olmo 3 Base

Tables 2 and 3 compare Olmo 3 Base with leading open base models, demonstrating both the effectiveness of our evaluation design and Olmo 3 Base 's strong performance across a broad set of capabilities.

Olmo 3 Base is the best fully open model at 32B parameter, outperforming Stanford Marin 32B. For the 32B scale, in Math and Code performance we see that Olmo 3 Base soundly outperforms other fully open models and remains closely on par with the most open weight models. In MCQA performance Olmo 3 Base is on par with other open models in Non-STEM areas, and with fully open models in STEM areas. Finally, in GenQA tasks, Olmo 3 Base outperforms most of the listed open models. As for the 7B scale, Olmo 3 Base

---

[2]We ran on 128 nodes with 8× NVIDIA H100 (80GB HBM3) per node, connected via TCPXO (200 Gbps/GPU). We used Hybrid-Sharded Data Parallelism (HSDP) via PyTorch FSDP2 (Zhao et al., 2023), with 8-way sharding so each node hosted a single model replica. Communication-intensive collectives were therefore restricted to within-node, improving efficiency.

[3]Same 8-way HSDP configuration but on 16 identical nodes.

[4]32 nodes with 8-way context parallelism, 32-way data parallelism, and activation checkpointing.

	Fully-open Models				Open-weight Models					
	Olmo 3 32B	Marin 32B	Apertus 70B	OLMo 2 32B	Qwen 2.5 32B	Gemma 3 27B	Mistral 3.1 24B	Seed 36B	Gemma 2 27B	Llama 3.1 70B
OlmoBaseEval Math	61.6	49.3	39.7	53.9	64.7	63.2	59.5	15.3	57.5	62.0
GSM8k	80.5	69.1	63.0	77.6	81.1	81.3	79.3	26.9	76.3	81.2
GSM Symbolic	61.0	42.0	38.6	53.1	56.2	61.2	59.1	10.3	57.3	64.6
MATH	43.4	36.8	17.4	31.0	56.7	47.0	40.1	8.7	38.8	40.2
OlmoBaseEval Code	40.0	30.8	23.3	20.5	48.3	41.6	42.4	54.9	41.0	36.3
BigCodeBench	43.9	34.5	24.0	22.2	48.1	44.0	46.4	50.7	43.4	43.4
HumanEval	66.5	52.3	32.5	29.4	65.6	62.1	65.5	71.3	57.5	57.4
DeepSeek LeetCode	1.9	1.3	1.2	0.8	8.0	5.8	0.1	13.0	4.7	0.2
DS 1000	29.7	26.3	17.8	20.4	43.3	34.3	36.3	44.0	29.7	29.5
MBPP	60.2	52.1	37.6	37.1	69.8	60.0	61.9	72.0	61.7	55.5
MultiPL HumanEval	35.9	18.5	18.4	10.5	49.7	37.7	39.0	69.2	40.3	32.2
MultiPL MBPPP	41.8	30.5	31.3	23.2	53.6	47.2	47.7	63.8	49.7	35.9
OlmoBaseEval MC$_{STEM}$	74.5	75.9	70.0	75.3	82.2	80.2	81.5	83.4	75.6	80.1
ARC MC	94.7	93.4	90.7	94.4	97.0	95.8	96.2	97.3	94.1	95.2
MMLU STEM	70.8	68.4	57.8	64.7	79.7	74.9	76.1	82.8	65.8	70.0
MedMCQA MC	57.6	61.8	55.9	60.2	68.8	64.7	68.8	69.6	61.8	67.8
MedQA MC	53.8	60.8	52.4	62.2	68.4	68.7	70.4	70.1	61.0	72.3
SciQ MC	95.5	95.1	93.3	95.1	97.1	96.8	96.3	97.1	95.1	95.4
OlmoBaseEval MC$_{Non-STEM}$	85.6	84.5	78.5	84.2	89.3	86.7	87.9	89.0	83.2	86.1
MMLU Humanities	78.3	78.9	74.1	79.7	85.0	80.5	82.7	85.7	79.3	83.4
MMLU Social Sci.	83.9	83.7	79.2	84.5	88.4	86.2	88.6	90.1	85.8	87.4
MMLU Other	75.1	75.4	70.1	75.6	81.2	80.2	81.9	82.4	76.9	79.4
CSQA MC	82.3	80.1	76.9	81.2	89.9	79.0	80.5	81.1	78.1	79.0
PiQA MC	85.6	90.5	79.0	87.7	93.3	90.3	91.0	92.5	89.0	91.5
SocialIQA MC	83.9	82.4	79.3	82.3	86.6	81.2	81.0	84.9	81.0	83.5
CoQA Gen2MC MC	96.4	93.9	87.5	94.4	96.8	95.8	94.9	96.9	94.3	95.1
DROP Gen2MC MC	87.2	71.0	56.5	68.6	86.6	84.6	86.5	90.1	66.6	70.3
Jeopardy Gen2MC MC	92.3	95.3	93.2	96.6	97.0	95.9	97.2	96.2	92.0	97.1
NaturalQs Gen2MC MC	78.0	81.0	71.9	78.6	79.9	82.0	84.6	81.4	74.5	82.4
SQuAD Gen2MC MC	98.2	97.6	95.7	97.4	97.9	97.7	97.9	98.1	97.5	97.7
OlmoBaseEval GenQA	79.8	80.3	75.0	79.1	68.5	73.5	78.0	76.0	72.9	81.6
HellaSwag RC	84.8	87.2	84.5	87.5	86.3	86.0	86.2	84.8	86.7	88.4
Winogrande RC	90.3	90.5	87.7	89.4	87.5	91.3	90.8	89.3	90.8	91.7
Lambada	75.7	76.7	74.8	77.0	76.2	77.5	79.3	76.1	76.9	79.6
Basic Skills	93.5	91.1	87.5	88.7	94.2	94.9	91.9	96.0	93.2	92.4
DROP	81.0	76.5	56.3	76.3	53.7	75.9	74.9	76.1	73.2	78.3
Jeopardy	75.3	80.5	77.2	79.1	74.0	82.1	80.3	77.4	80.7	84.0
NaturalQs	48.7	55.1	43.1	51.4	39.3	49.2	45.1	30.7	47.1	53.1
SQuAD	94.5	94.4	90.7	94.0	64.9	92.4	92.6	89.1	93.0	92.9
CoQA	74.1	70.7	72.8	68.7	40.4	12.4	61.1	64.4	14.9	73.9
OlmoBaseEval HeldOut	-	-	-	-	-	-	-	-	-	-
BBH	77.6	70.1	58.8	64.6	81.1	77.4	81.4	85.0	74.8	80.8
MMLU Pro MC	49.6	48.1	39.6	46.9	61.1	53.1	58.9	62.2	47.6	50.4
Deepmind Math	30.1	26.7	20.1	22.0	40.7	30.4	35.3	31.3	27.6	40.3
LBPP	21.7	17.3	8.1	8.2	40.3	17.7	30.3	42.6	19.7	11.8

Table 2 Results comparing Olmo 3 Base 32B to other base models using the OlmoBaseEval Main suite (details in §3.3). Olmo 3 was not evaluated on held-out benchmarks prior to release.

dramatically outperforms other fully open models in Math and Code performance, and is outperformed only by Qwen models in Math, and Qwen and Nemotron in Code. In MCQA, Olmo 3 Base is on par with the strongest fully open models in both STEM and Non-STEM areas. Finally, in GenQA tasks Olmo 3 Base outperforms all but Marin among listed fully open models, and outperforms all but the larger Gemma 2 9B and Llama3.1 8B among listed open weight models.

3.2 Modeling and Architecture

Olmo 3 modeling and training largely follows that of OLMo 2. We focus this section on the key differences and refer to the appendix for further details.

Architecture. We adopt a decoder-only transformer architecture based on Vaswani et al. (2017). Details of the architecture are presented in Table 30 in Appendix A.1. Compared to OLMo 2:

- We train with a context window of 8192 instead of 4096 tokens during pretraining and midtraining stages.
- To support scalable pretraining at longer sequence lengths, and to keep inference costs manageable, we

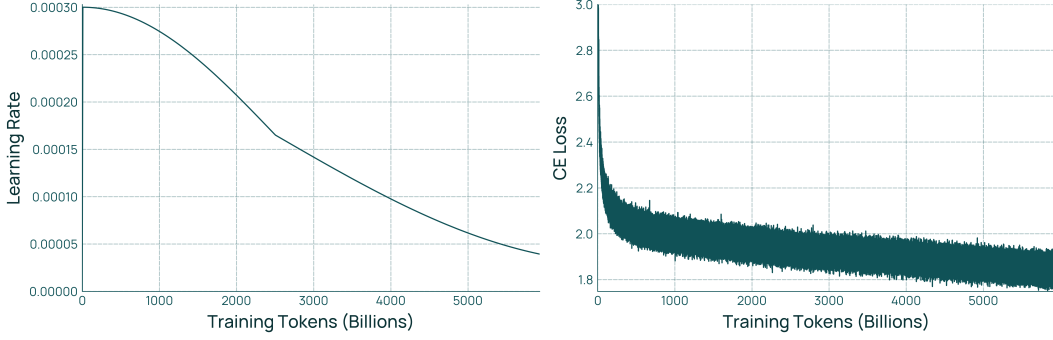|	Fully-open Models				Open-weight Models						
	Olmo 3 7B	Marin 8B	Apertus 8B	OLMo 2 7B	Qwen3 8B	Nemo. Nano 9B	Gemma 2 9B	Qwen 2.5 7B	Llama 3.1 8B	Granite 3.3 8B	MiMo 7B
OlmoBaseEval Math	54.7	39.6	29.2	41.7	67.2	49.8	48.8	60.7	36.9	41.5	54.3
GSM8k	75.5	60.9	48.2	67.1	84.5	82.3	68.5	79.9	56.4	61.0	74.3
GSM Symbolic	48.6	33.6	26.3	38.8	65.4	62.7	45.1	56.2	35.1	35.5	53.3
MATH	40.0	24.3	13.1	19.1	51.6	4.5	32.9	45.9	19.2	27.9	35.2
OlmoBaseEval Code	30.7	21.4	19.0	10.4	46.1	43.1	30.2	41.0	21.2	18.0	35.7
BigCodeBench	34.1	21.5	20.9	8.8	42.5	43.2	30.9	39.7	30.7	0.4	38.3
HumanEval	49.1	31.6	21.6	16.3	71.7	71.7	40.0	66.1	40.4	0.0	57.0
DeepSeek LeetCode	1.4	0.5	0.6	0.2	8.3	6.8	1.9	5.1	0.1	0.0	1.2
DS 1000	20.2	16.5	11.8	10.1	33.1	30.3	23.4	35.2	22.2	22.6	28.1
MBPP	43.6	36.5	33.5	21.2	66.2	62.3	49.1	55.4	12.1	48.5	48.3
MultiPL HumanEval	28.7	15.6	15.5	4.2	52.3	40.0	27.9	40.3	14.5	22.3	34.5
MultiPL MBPPP	38.2	27.6	29.2	12.2	48.4	47.5	38.2	45.4	28.3	32.3	42.5
OlmoBaseEval MC$_{\text{STEM}}$	66.4	68.1	66.3	64.6	78.8	73.5	72.8	74.7	69.0	65.0	71.6
ARC MC	89.2	89.2	87.9	85.7	95.4	94.1	92.7	93.4	86.4	86.2	91.7
MMLU STEM	59.7	58.1	52.4	53.2	76.7	71.1	62.8	67.6	55.7	55.6	63.5
MedMCQA MC	48.3	52.7	51.7	49.2	63.5	54.5	58.9	60.3	56.5	49.6	56.2
MedQA MC	41.8	47.3	47.6	43.8	62.1	53.5	55.4	56.6	53.7	43.0	53.0
SciQ MC	92.8	93.2	91.9	90.9	96.1	94.3	94.4	95.4	92.7	90.8	93.5
OlmoBaseEval MC$_{\text{Non-STEM}}$	78.2	78.8	74.2	75.2	84.8	81.3	81.3	82.9	76.1	76.9	80.5
MMLU Humanities	68.9	71.4	67.8	67.9	78.6	78.0	74.5	76.2	70.1	67.6	73.6
MMLU Social Sci.	75.0	77.4	74.7	73.1	84.8	82.2	82.9	83.0	75.5	71.8	80.8
MMLU Other	66.9	68.3	66.1	65.2	76.8	73.8	74.2	74.4	69.1	64.5	72.7
CSQA MC	75.3	75.3	72.1	72.0	84.1	74.4	75.3	85.0	72.9	82.3	76.1
PiQA MC	80.2	85.7	80.5	80.1	89.9	86.0	85.7	88.5	78.3	81.5	87.2
SocialIQA MC	80.3	79.8	76.3	77.5	83.3	78.7	80.3	82.9	77.0	83.1	80.7
CoQA Gen2MC MC	92.5	86.2	82.8	85.0	93.7	92.2	92.7	93.5	89.9	87.6	91.4
DROP Gen2MC MC	67.3	63.7	47.5	55.6	78.3	70.0	65.8	69.1	53.3	55.0	64.1
Jeopardy Gen2MC MC	86.9	90.8	90.3	89.5	92.3	90.7	92.8	92.1	88.9	88.4	89.5
NaturalQs Gen2MC MC	69.4	71.5	66.7	66.3	74.1	71.1	72.5	70.5	68.0	69.2	72.2
SQuAD Gen2MC MC	96.9	96.5	91.3	95.3	97.5	97.4	97.3	96.4	94.4	94.5	96.7
OlmoBaseEval GenQA	72.5	75.9	69.0	72.4	71.1	71.8	75.6	67.5	73.1	67.8	71.4
HellaSwag RC	77.7	84.0	81.0	82.2	80.5	80.2	81.8	81.0	81.5	83.7	80.6
Winogrande RC	85.7	88.6	85.8	87.4	86.4	86.2	88.8	86.0	87.3	89.4	86.5
Lambada	68.9	73.9	70.9	70.5	73.0	67.9	76.3	70.3	75.5	76.0	73.1
Basic Skills	89.5	85.6	83.8	82.2	93.5	91.4	89.3	91.4	88.0	88.7	89.7
DROP	71.5	73.0	37.1	61.5	57.2	71.4	68.2	56.7	59.5	38.4	69.3
Jeopardy	60.4	72.7	70.1	70.8	65.1	64.9	75.1	63.0	70.9	69.7	65.6
NaturalQs	32.6	42.6	35.0	37.4	33.8	31.2	40.4	31.2	36.7	37.0	33.1
SQuAD	93.5	93.4	89.6	91.5	89.2	92.3	88.8	87.0	89.2	89.6	90.3
CoQA	72.8	69.5	67.4	68.3	61.6	60.4	71.5	40.5	69.0	37.8	54.4
OlmoBaseEval HeldOut	-	-	-	-	-	-	-	-	-	-	-
BBH	63.5	55.6	48.1	49.6	76.5	77.0	68.8	54.7	63.0	61.5	75.1
MMLU Pro MC	37.3	38.8	33.9	33.1	50.3	50.2	44.7	48.1	37.4	33.9	44.3
Deepmind Math	23.7	20.2	17.1	16.3	47.7	31.4	23.0	32.8	24.1	32.2	25.4
LBPP	17.1	5.8	7.1	3.1	25.7	31.7	12.4	22.1	9.1	18.5	21.5

Table 3 Results comparing Olmo 3 Base 7B to other base models using the OlmoBaseEval Main suite (details in §3.3). Olmo 3 was not evaluated on held-out benchmarks prior to release.

introduce a sliding window attention (SWA) pattern (Beltagy et al., 2020) in which each token can attend to previous tokens in a window of size 4096. We added SWA at three out of every four layers, and ensure that the last layer always uses full attention.
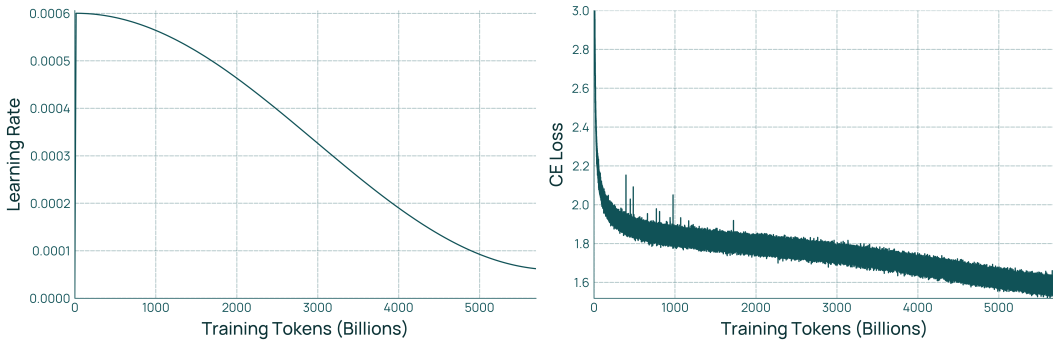
- During long-context extension, we extend our context window from 8k to 64k over 50B / 100B tokens using YaRN (Peng et al., 2023). For more details, check Section 3.6.4.

Training. Olmo 3 Base was trained using the **OLMo-Core** codebase. Using this codebase, we can train a 7B model at 7700 tokens per second per GPU, and a 32B at 1900 tokens per second per GPU, at a sequence length of 8192, and with `bfloat16` precision throughout. This is equivalent to roughly 43% and 41% MFU respectively. We achieve this performance by relying on PyTorch's built-in `torch.compile()`, custom kernels for operations such as attention and language modeling head, asynchronous and batched gathering of metrics, and asynchronous writing of checkpoints, and others. OLMo-core supports pretraining, mid-training, and SFT, along with auxiliary tools for checkpoint conversion to and from Huggingface format, and tools for merging model checkpoints. Support for DPO and RL is planned but incomplete. Further details and the code itself are available at `github.com/allenai/OLMo-core`.

Parameters for training Olmo 3 Base-7B and 32B are presented in Tables 31 and 32 in Appendix A.1. As in

Figure 3 Learning rate schedule and loss for OLMO 3 BASE 7B. The first half of the learning rate schedule is a cosine schedule over 5T tokens. We stretch the second half of the schedule to reach a target length of one epoch (5.93T tokens). Warm-up is 2000 steps, the peak learning rate is 3×10^{-4}, and the final learning rate is 10% of the peak LR.

Figure 4 Learning rate schedule and loss for OLMO 3 BASE 32B. The learning rate schedule is a cosine schedule over one epoch (5.93T tokens), truncated at 5.5T tokens. Warm-up is 2000 steps, and the peak learning rate is 6×10^{-4}. The schedule targets a final learning rate of 10% of the peak. Due to the truncation, the real final learning rate is 6.210×10^{-5}. Unintuitively, the learning rate for the 32B is higher than the learning rate for the 7B, but this is somewhat compensated for by the larger batch size of the 32B (8M tokens vs. 4M tokens per batch).
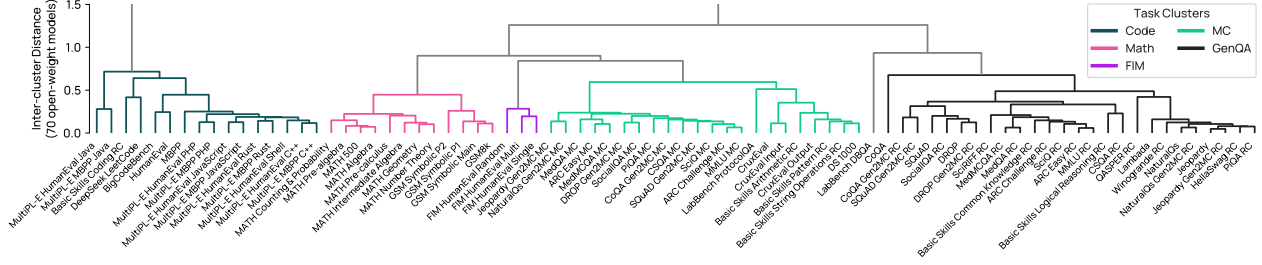
OLMO 2, we train in stages defined by data curriculum and learning rate schedule (more details in Appendix Tables 31 and 32).

Tokenizer. We process data for each stage using the same tokenizer as OLMO 2, which is derived from OpenAI's `cl100k` (OpenAI, 2023a,b).

3.3 Experimental Design and Evaluation

Model development requires many iterative data and training decisions. However, benchmarks are not perfect decision-making tools: different evaluations are only sensitive for making development decisions across specific ranges of scale and capability (Magnusson et al., 2025). Models trained at small compute scales are known to exhibit random-chance performance on math, code and multiple-choice question answering (MCQA) tasks (Wei et al., 2022; Gu et al., 2024b) and benchmark noise can reduce the ability to trust small differences in scores (Heineman et al., 2025). To address these problems, we develop **OLMOBASEEVAL**, a collection of benchmark suites to support decision-making during base model development. OLMOBASEEVAL features the following improvements:

- We aggregate scores over **task clusters** that group benchmarks by assessed capability (§3.3.1),
- We develop **proxy metrics** for evaluating small-scale models by identifying when capabilities "emerge" during training (§3.3.2), and
- We improve the overall **signal-to-noise ratio** by evaluating on more examples from noisy tasks or even

Figure 5 Task clustering for OLMOBASEEVAL. Using a set of 23K benchmark results, the clustering method iteratively merges tasks which rank models similarly, until arriving at a stop condition. To arrive at OLMOBASEEVAL, we move tasks in the same format into the same cluster and split MC into STEM and Non-STEM tasks.

removing them entirely (§3.3.3).

We started by targeting a high *coverage* of capabilities, we selected benchmarks to prioritize science knowledge, medical/lab knowledge, math and code tasks. Because our data interventions are targeted to a core capability rather than a specific benchmark (e.g, "Code" rather than "DS-1000"), we group tasks into *clusters*, where we expect the benchmarks within a cluster to behave similarly to particular data changes. To handle abilities where models trained to using small compute budgets (e.g. up our largest experiment scale of 1B parameters at 100B tokens), we perform a *scaling analysis* to determine which benchmarks show signal at a small scale, and find proxy metrics which we use to make decisions. Finally, we analyzed the *signal-to-noise ratio* of each benchmark – we selected benchmark metrics to improve SNR, removed benchmarks that were too noisy for making decisions, and moved benchmarks out of the average if the noise of one particular benchmark dominated the aggregate scores.
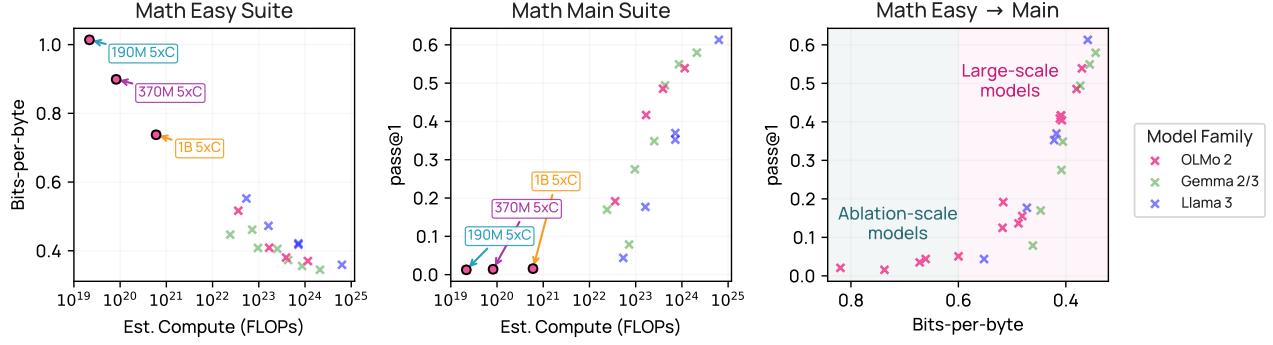
3.3.1 Clustering Tasks

To handle the large number of tasks, we cluster similar tasks into macro-averages. We aim for task clusters to match the granularity at which we perform data interventions, and for tasks within each cluster to behave similarly. Our clustering procedure requires a process to determine the similarity of two evaluations—we do this by collecting a pool of 23K benchmark scores from 70 external, open-weight models.

Using our dataset of evaluation results, we assume that two benchmarks evaluate similar constructs if they rank models similarly. We perform hierarchical clustering using Ward's variance-minimization (Ward Jr, 1963), which iteratively merges evaluation scores to minimize the variance of scores between benchmarks within a cluster. Figure 5 shows the result of the clustering procedure, where we manually select a threshold to balance the amount and granularity of clusters. Importantly, we do not use the exact result of the clustering procedure – we manually move a few tasks to ensure the format of the task is the same within each cluster (e.g. tasks requiring code execution all occur in the same cluster). The resulting task clusters are: MC_{STEM}, $\text{MC}_{\text{Non-STEM}}$, GenQA, Math, Code and Code FIM.
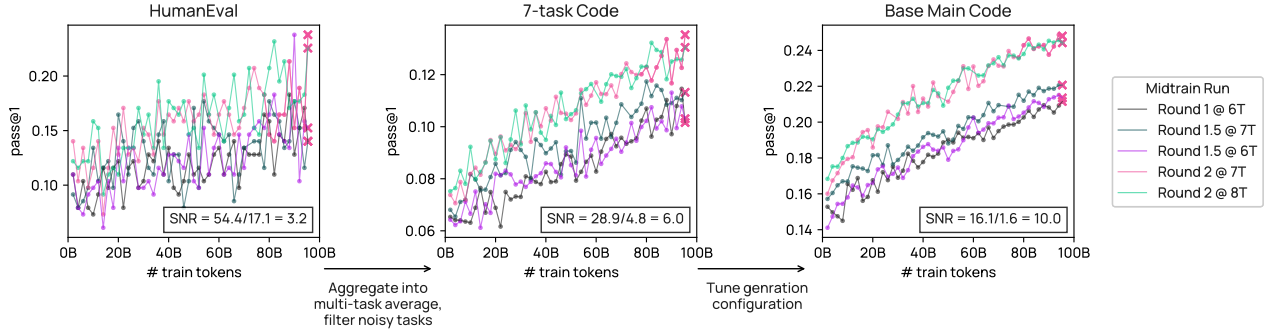
3.3.2 Scaling analysis

We evaluate open-weight models across compute scales from 10^{18} to 10^{25} training FLOPs to determine the compute scale at which particular metrics and tasks are useful for development decisions. On some evaluation benchmarks it is too difficult to see signal when training models at small scales (Wei et al., 2022), and other benchmarks 'saturate' near the labeling error of the benchmark (Vendrow et al., 2025). However, while many tasks appear emergent, continuous proxy metrics have been shown to be a better decision making tool for model performance before we exit the noise floor (Schaeffer et al., 2023; Huang et al., 2024b; Magnusson et al., 2025). We propose a Base Easy task suite which measures bits-per-byte over tasks from the Base Main suite which have gold labels or human-written answers, calculated as the negative log-likelihood of the answer divided by the number of UTF-8 bytes in the answer string, as described in Gao et al. (2020).

We evaluate on the suite of 25 OLMo 2 scaling law models from (Bhagia et al., 2024) to understand the scaling behavior in the low-compute regime, and 70 open-weight models to understand scaling behavior in the

Figure 6 Scaling analysis on the OLMoBASEEVAL Math suite. We use the OLMo 2 scaling models (Bhagia et al., 2024) to find benchmarks and metrics that showed signal for small scale models (left and center). Then, we use the small scale OLMoBASEEVAL Easy suite as a proxy-metric for making data decisions.

Figure 7 OLMoBASEEVAL signal-to-noise analysis on the code multi-task average using intermediate checkpoints from midtraining. First, we aggregate into multi-task averages and remove tasks with high noise, such as CruxEval (left → center). Then, we tuned generation parameters to improve SNR, e.g. by increasing the n in pass@k (center → right).

high-compute regime. Figure 6 shows the scaling behavior for our resulting Base Main benchmarks. For each task family, the **Base Easy** task suite showed signal at the small data ablation scale, and the **Base Main** task suites were not saturated at the large scale, leaving headroom for data experiments in midtraining.

3.3.3 Signal-to-Noise Analysis

When reporting a macro-average, we aimed to exclude tasks from each cluster that were too noisy to be helpful for development. We calculate the signal-to-noise ratio of each benchmark following the method from Heineman et al. (2025), where we evaluate the final 50 checkpoints of OLMo 2 13B training, and 10 external base models trained at roughly the same compute scale ($4 \cdot 10^{23}$ FLOPs). From our findings, we transition from using 1K instance subsets to full evaluation sets when available. We remove some benchmarks from our evaluation suite entirely, particularly binary benchmarks such as BoolQ (Clark et al., 2019), as we found that models usually oscillate between predicting the majority and minority class.

We repeat the same analysis for midtraining, instead using intermediate checkpoints from 5 preliminary pretraining runs. One important finding was to separate some benchmarks from the macro-average, like CruxEval (Gu et al., 2024a), which measures a relevant and unique capability (code input/output prediction), but would introduce too much noise into the macro-average. We show an example of the SNR of three individual benchmarks compared to the base main task averages across intermediate checkpoints during midtraining in Figure 7.

3.3.4 OlmoBaseEval

The resulting OLMoBASEEVAL consists of a **Base Easy** suite for making development decisions using small compute budgets (e.g. less than 1B parameters) and a **Base Main** suite for development decisions for the

final pretraining run and midtraining. We provide detail on the **Chat** suite later in §4.1. OLMOBASEEVAL contains 43 tasks, 4x more benchmarks than OLMo 2 including tracking math and code benchmarks in pretraining. To prevent overfitting on the development suite, we include a **Held-out** set of 4 benchmarks – MMLU Pro, Deepmind Math, LBPP and BBH – each benchmark matching one broad capability we targeted during pretraining.

This includes four new benchmarks: **BasicSkills**, a set of 6 tasks to isolate the development of skills during pretraining (e.g. basic arithmetic, reasoning, and coding); **Gen2MC**, a multiple-choice version of 5 short-form generative tasks; and **MT MBPP**, a translated BPB set for MBPP in 17 code languages; **Masked Perplexity**, a new evaluation method by applying token masking and calculating perplexity only on tokens that are difficult to learn. We evaluate with masked perplexity using UltraChat and WildChat, which provides a wide coverage of real user interaction evaluation in pretraining. Additional design and implementation details for OLMOBASEEVAL are included in Appendix A.3.

3.4 Stage 1: Pretraining

Source	Type	9T Pool		6T Mix		150B Mix	
		Tokens	Docs	Tokens	Docs	Tokens	Docs
Common Crawl	Web pages	8.14T	9.67B	4.51T (76.1%)	3.15B	121B (76.9%)	84.5M
olmOCR Science PDFs	Academic documents	972B	101M	805B (13.6%)	83.8M	19.9B (12.6%)	2.25M
StackEdu (Rebalanced)	GitHub code	137B	167M	409B (6.89%)	526M	11.1B (7.06%)	14.3M
arXiv	Papers with LaTeX	21.4B	3.95M	50.8B (0.86%)	9.10M	1.29B (0.82%)	247K
FineMath 3+	Math web pages	34.1B	21.4M	152B (2.56%)	95.5M	4.10B (2.60%)	2.57M
Wikipedia & Wikibooks	Encyclopedic	3.69B	6.67M	2.51B (0.04%)	4.24M	64.6M (0.04%)	119K
Total		**9.31T**	**9.97B**	**5.93T (100%)**	**3.87B**	**157B (100%)**	**104M**

Table 4 Composition of DOLMA 3 including our 9T pool of data, the 6T mix we used for final model training, and the 150B mix we used for experimentation.

We first train OLMO 3 BASE on **Dolma 3 Mix**, our 6T token pretraining data mix. While DOLMA 3 MIX is comprised of largely the same types of data sources used in other open pretraining recipes (Soldaini et al., 2024; Bakouch et al., 2025; OLMo et al., 2024), we demonstrate three key novelties:
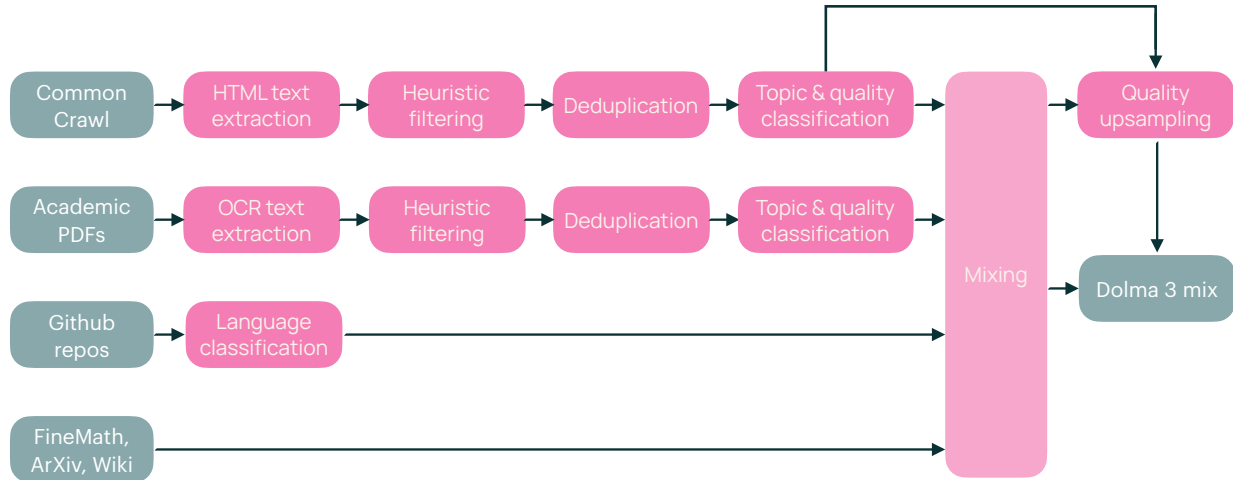
- New tooling for fast and scalable **global deduplication** at the trillion-token scale,
- Two new methods for optimizing selection of training tokens: **token-constrained mixing** and **quality-aware upsampling**,
- A novel source of academic PDFs—**olmOCR Science PDFs**—converted to linearized plain text using OLMOCR (§3.4.2) (Poznanski et al., 2025a) .

Table 4 summarizes our data sources, pool sizes, and final training mix.[5] As developing a base model is the most compute-intensive part of our development process, requiring training over trillions of tokens and consuming over 90% of overall compute, we adhere to two major principles to guide our data strategy:

1. We consider a source of data for pretraining if it has potential to yield enough tokens to impact model capabilities at pretraining scale. Valuable data sources that are small may not be impactful in pretraining and are better reserved for midtraining.

2. While we embrace exploration of structured "task" data (e.g. QA pairs, chat instances) for training base models, we reserve their use only for later stages of midtraining (§3.5) and long-context extension (§3.6). Task data often does not meet the pool size needed to impact our pretraining phase, even with synthetic generation, and task data also tends to have an outsized impact on evaluation results, potentially confounding data ablations for other sources.

---

[5]The training mixes that we release represent reconstructions of the data sampled during our actual training runs. Tokens included in these reconstructions represent all of the tokens trained on for the training run, while included documents represent a union of all unique documents that contributed at least one token during training.

Figure 11 summarizes the pipeline steps for creating DOLMA 3 MIX pretraining data. We describe them in more detail in the remainder of this section.

Figure 8 Data curation flow of sources in DOLMA 3 MIX.

3.4.1 Preparing our Web Data Pool

We took the following steps to curate pretraining data from CommonCrawl (Common Crawl Foundation), which constituted the majority of our pretraining corpus.

Text extraction We started with 104 dumps from the CommonCrawl corpus, with a cutoff date of December 31, 2024. Following DCLM (Li et al., 2024a), we removed HTML artifacts and extracted the semantic text from WARC files using Resiliparse (Bevendorff et al., 2018). Where applicable, we directly leveraged the raw Resiliparse-extracted data from DCLM-pool (Li et al., 2024a)[6] and applied Resiliparse extraction on dumps not contained with the DCLM-pool.

Heuristic Filtering We applied a pipeline of heuristic filtering steps to prune our initial collection of 252.6B documents to a size amenable for pretraining. Our process closely followed that of DCLM (Li et al., 2024a) with minor modifications to improve data quality and computational efficiency. We first applied URL filtering to remove spam and adult-content from an expanded blocklist. We then removed documents that were either too short or too long, followed by filtering documents that contained excessive symbols or insufficient quantities of alphabetic characters. Next we removed documents containing large amounts of internal repetition and applied filtering to remove common spam phrases, fully removing any documents that were eviscerated by such filtering. We then used a fastText classifier to identify the language of each document, keeping only documents that contained English text. As a final step, we applied sentence-level heuristics from Madlad400 (Li et al., 2024a). In aggregate, this process reduced the size of our data pool by 84.6%, yielding a corpus of 38.8B documents. More details are provided in §A.1.

Deduplication The web data collected from CommonCrawl naturally contains an abundance of duplicated documents. This duplication arises from repeated crawls of the same website, near-copies of documents

---

[6] https://data.commoncrawl.org/contrib/datacomp/DCLM-pool/index.html

appearing across multiple web pages, and highly-repeated boilerplate text. Our deduplication strategy was motivated by three observations from prior work: i) deduplication generally leads to more token-efficient training (Lee et al., 2022); ii) duplicate count serves as a weak signal of data quality, with higher duplicate counts indicating higher quality (Fang et al., 2025a); iii) repeating documents more than a handful of times provides rapidly diminishing returns (Muennighoff et al., 2025a).

Given these observations, we designed our deduplication strategy to enable a future quality-based upsampling step (Section 3.4.4). We aggressively deduplicate our dataset at multiple granularities, targeting the removal of exact replicas, near-duplicates, and repeated filler text. While this necessarily discards the quality signal from duplicate counts, it produces a clean base dataset from which we can later selectively reintroduce repetition for high-quality documents. Our goal is a final dataset with minimal repetition overall, with any duplication concentrated in high-quality data. We implemented our deduplication procedure in three distinct stages:

1. **Exact Deduplication:** We applied global deduplication based on document text hashes to remove all exact copies. This step identified 67% of the pool as duplicates, reducing the dataset from 38.7B to 12.8B documents.

2. **Fuzzy Deduplication:** We applied MinHash-based deduplication to identify and remove near-identical documents, such as documents copied across multiple domains that differ only in headers or footers. We partitioned the dataset into 32 shards, ran MinHash deduplication on each shard, then performed exhaustive pairwise Jaccard similarity checks within each identified cluster. From each cluster, we retained the most recent document by crawl date. This procedure identified 23% of the pool as duplicates, yielding 9.8B documents.

3. **Substring Deduplication:** The previous steps removed whole duplicate documents but did not address repeated content within individual documents. Many documents contain substantial boilerplate text or HTML artifacts (e.g., headers and footers) of limited training value. To remove these repeated substrings, we applied a novel fuzzy suffix-array-based deduplication procedure. We partitioned the dataset into 57 shards and applied this procedure to each, marking any substring of 500 or more bytes that occurred multiple times. Unlike previous suffix-array methods, we preserved at least one occurrence of each repeated substring in the corpus. We then merged the intervals marking repeated substrings to also remove short substrings sandwiched between longer repeated segments. This procedure removed 14% of text bytes, yielding 9.7B documents totaling 36.5T bytes of uncompressed text.

This three-stage procedure reduced the web corpus from 38.7B to 9.7B documents—a 75% reduction in document count. The resulting aggressively deduplicated dataset can then be partitioned by topic and quality and controllably upsampled for training.

To scale our deduplication strategy, we develop the Duplodocus tool[7], a native-rust toolkit for large-scale distributed execution of both hash-based exact deduplication and MinHash fuzzy deduplication.

Topic and Quality Classification   We used our WebOrganizer tool (Wettig et al., 2025a) to partition the deduplicated corpus into 24 topics (e.g., "Science and Technology", "Politics"). We also trained and applied a fastText-based quality classifier to assign each document a quality score. Following DCLM (Li et al., 2024a), we used OpenHermes-2.5 (Teknium, 2023) and ELI5 (Fan et al., 2019) as positive training examples, supplemented with UltraChat-200k (Ding et al., 2023) and WildChat-1M (Zhao et al., 2024a). Negative training examples consisted of 30GB sampled from DCLM-RefinedWeb.

We applied both the topic and quality classifiers to the full deduplicated corpus in order to partition the dataset. Documents were first partitioned by topic, then within each topic partition, we computed quality score percentiles and subdivided documents into vigintile buckets (5-percentile intervals). This two-stage partitioning yielded 480 disjoint subsets (24 topics × 20 quality tiers), enabling fine-grained control over the topic and quality distribution of our pretraining mixture.

Final Web Data Pool   The above steps resulted in an 8T-token pool of annotated data, partitioned into buckets according to topic and text quality. This pool serves as the foundation for our pretraining mixture, though

---

additional processing is required to construct the final training data. Specifically, we apply quality-based filtering and topic reweighting to generate a balanced, high-quality mixture, as discussed in §3.4.4.

3.4.2 Preparing our PDF Data Pool

We curated a novel dataset of academic PDFs, replacing our previous use of peS2o (Soldaini and Lo, 2023). These documents were crawled "politely": with a disclosed user agent, adhering to `robots.txt`, and respecting paywalls. The crawler was seeded with a focus on academic sites and paper repositories. We processed all PDFs using olmOCR (Poznanski et al., 2025a). Ultimately this crawl generated a collection of 238M unique PDF documents with a cutoff date of December 2024.

OlmOCR Text Extraction To convert PDFs to a format usable by our trainer, we applied pre-filtering and text extraction. If a document contained born-digital text, we used the Lingua language detector to retain only English documents and removed documents where spam or SEO-optimization keywords exceeded 0.4% of total words. We then extracted text using olmOCR (Poznanski et al., 2025a) (versions 0.1.49-0.1.53). When olmOCR failed, we used Poppler's `pdftotext` as a fallback; documents requiring this fallback for more than 1 in 250 pages were excluded from the corpus. This yielded a dataset of 160M PDF documents.

Deduplication We then identified and removed any fuzzy-duplicates using a MinHash algorithm. This differed slightly from MinHash step applied to the web text corpus in §3.4.1: we used the MinHash parameters as in FineWeb (Penedo et al., 2024), which targeted document pairs with at least 75% similarity; and we omitted an exhaustive pairwise Jaccard similarity check. After this deduplication step, we were left with a corpus of 156M documents for a removal rate of 2.3%.

PII Filtering Next we removed documents containing PII from the pool of PDFs. We wanted to remove documents that contained sensitive standalone PII, such as government IDs and login information. We also wanted to remove documents that linked biographical, medical, location, employment, or educational information to a specific individual, but specifically wanted this filtering to be context aware. Some documents such as public meeting minutes, contain PII identifiers such as names linked to addresses, but are meant for public dissemination, and we did not want to remove these. After experimenting with use of human annotators to identify and remove PII, we developed a multi-stage model-based PII-identification pipeline.

First we classified documents using a prompt to `gemma-3-12B` (Gemma 3 Team, 2025) on the first page of each document to determine if they contained any sensitive standalone PII, or linked sensitive information to an individual. Next, we used `gemma-3-4B` on the first 5,000 characters of each document to arrive at a set of flags describing the type of document. From these classification results we developed a set of rules to identify which types of documents containing PII should be publicly available and which should be filtered. Ultimately this removed 4.9% of the remaining pool and yielded a pool of 148M documents. See Poznanski et al. (2025a) for more details regarding PII removal.

Heuristic Filtering After PII removal, we applied a round of heuristic filtering to further remove low-quality documents. Filters applied in this step included checking for: non-English documents not originally caught by the Lingua filter; documents that were more than 30% tables; and documents that contained more than 20% numbers. Next we applied modifications that converted markdown tables to HTML and removed URL references. Finally, for legal purposes, we removed documents that did not contain permissive licenses. The combination of these filtration steps yielded a corpus of 108M documents. This corpus was then into partition into 24 topical buckets, according to the WebOrganizer topic classifier (Wettig et al., 2025b), and passed off to the mixing phase.

3.4.3 Preparing Code, Math, and other sources

Code For code data, we used StackEdu (Allal et al., 2025), an improved curation of GitHub repositories from the-stack-v2 dataset (Lozhkov et al., 2024) with additional filtering for educational programming content. We partitioned the data by programming language for subsequent mixing.

Math   As in OLMo 2, we include arXiv documents from the Proof-Pile-2 dataset (Azerbayev et al., 2023), which in turn are from RedPajama dataset (Together AI, 2023) and have a cutoff date of April 2023. We use this source primarily because it preserves the original LaTeX notation, enabling the model to learn both mathematical content and how to properly format it.

Furthermore, we replaced our previous use of OpenWebMath (Paster et al., 2023) with FineMath (Allal et al., 2025), a subset of Common Crawl documents that contain mathematical educational content and have been reprocessed to preserve proper mathematical notation. We include all documents that have a quality score of at least 3 (out of 4), according to the FineMath classifier. This data has a cutoff date of September 2024.

Other   Finally, we include the Wikipedia and Wikibooks sources from Dolma1.7 (Soldaini et al., 2024) as base sources of encyclopedic knowledge. These are both the "English" and "Simple" editions of Wikipedia and Wikibooks with a cutoff date of March 2023. These sources were processed using WikiExtractor (Attardi, 2015) to remove markup formatting, and all documents with 25 or fewer words were filtered out to exclude templated pages or pages that encountered XML parsing errors.
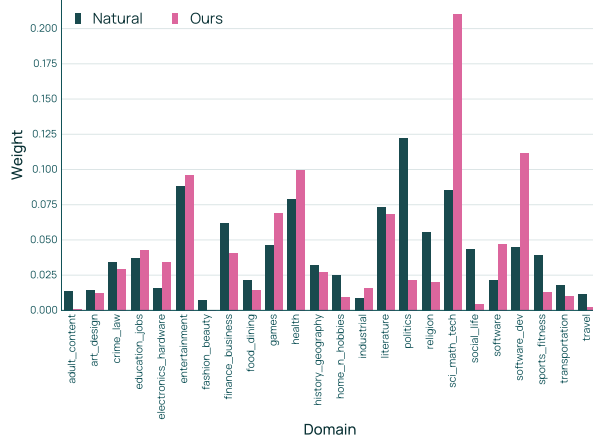
3.4.4   Sampling and Mixing over Data Pools

The data sources described above collectively provide over 9 trillion tokens of diverse text data. Transforming this collection into a training dataset requires a mixing and sampling pipeline to prescribe exactly how much of each source to include in a final training mix, and how much, if any, upsampling to apply to each source. We apply a mixing strategy that draws on swarm-based methods to train and evaluate many smaller proxy models, using these results to inform an optimal mix. Further, we apply a novel conditional mixing procedure to account for the fact that our data sources were being constantly refined and updated throughout the development cycle. In this section, we describe how we arrived at the mixing ratios for each source; for web text, we only optimize ratios at the WebOrganizer category level and apply quality-aware upsampling to obtain the final mix.

Constrained Data mixing   We applied data mixing across all pre-training sources, as well as across the WebOrganizer topics within the web data and PDF sources, and the StackEdu programming languages. Our mixing procedure (Chen et al., 2025a), consists of two components: a base procedure that constructs a high-quality mix over a fixed set of data domains, and a meta-procedure called conditional mixing that efficiently updates an existing mix when domains change. Together, these allow us to iteratively build an optimal mix and adapt to data refinements or additions without starting from scratch.
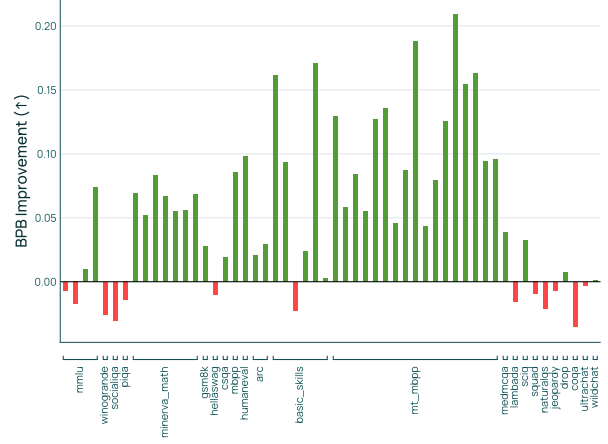
The base procedure follows a swarm-based approach inspired by RegMix, Data Mixing Laws, and CLIMB, (Liu et al., 2024a; Ye et al., 2025; Diao et al., 2025), and proceeds in three stages:

1. **Swarm construction**. We sample the space of possible mixes by training many small proxy models, each with a different mixing ratio. Specifically, we trained 30M-parameter models following the OLMo 3 architecture for 3B tokens (5x Chinchilla), sampling each mix from a Dirichlet distribution centered on the natural (no-mixing) distribution. As a rule of thumb, we launch a swarm of size 5x that of the number of domains. We then evaluate each proxy model on the Base Easy suite.

2. **Per-task regression**. Each proxy model provides a data point mapping mixture weights to task performance (measured in BPB) for each task. We fit a separate generalized linear model for each task, enabling us to predict how any candidate mix will perform.

3. **Mix optimization**. We find the mixture that minimizes the average task BPB, as predicted by the per-task regression models. Since we ultimately seek a corpus with a 6T token budget, and we avoid repeating any domain more than approximately $4 - 7$ times, this naturally imposes maximum ratio constraints on certain domains based on their available token counts. We solve this constrained optimization using a guided search initialized from a prior or natural distribution.

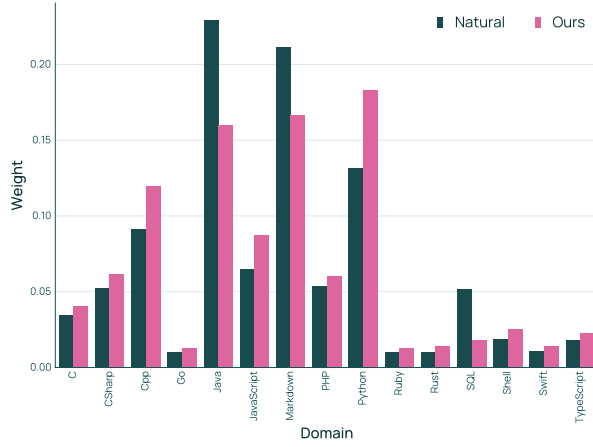The base procedure assumes fixed domains, but real preprocessing workflows evolve continuously as we refine filters, add domains, or discover and mitigate quality issues. Rather than recomputing an entire swarm each time domains change, we introduce a new procedure called conditional mixing to efficiently adapt the base method to an evolving data landscape. The key idea is to treat the existing optimized mix as a single virtual
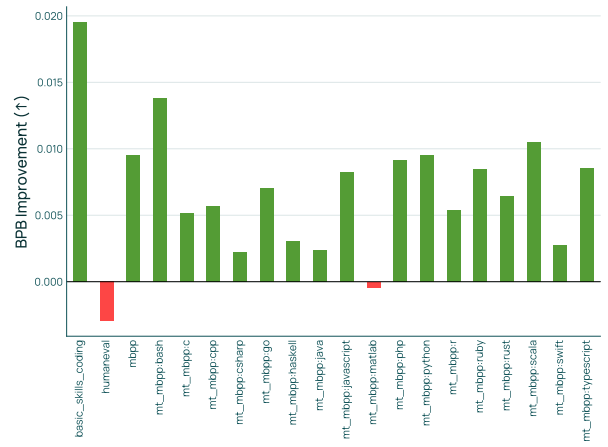
(a) DCLM partitioned by topic.

(b) Improvement when training over DCLM.

(c) StackEdu partitioned by programming language.

(d) Improvement when training over StackEdu.

Figure 9 **(a), (c)** Comparison of the natural distribution of data source in the DOLMA 3 pool versus our learned data mixture in DOLMA 3 MIX. **(b), (d)** The improvement on downstream evaluations resulting from training our data mix compared to the natural distribution.

domain with frozen mixing ratios, then re-run the base procedure over this virtual domain plus any new or modified domains. This effectively restricts the base mixing procedure to a lower-dimensional subspace of the mixture weight space, reducing swarm size and computational cost. Further details and justification of this procedure can be found in Chen et al. (2025a).

To construct the DOLMA 3 MIX weights, we performed three rounds of our conditional mixing procedure, with each stage building incrementally on frozen mixtures from prior stages. We first obtained optimized mixture weights over the 24 WebOrganizer categories within the `DCLM-Baseline mix`[8] as well as the source-level mix. Web text served as the starting point because it constitutes the largest data pool and because we used it to develop the base mixing methodology. Since the bespoke web data pool we crafted in Section REF was still under construction while performing these initial mixing rounds, we performed this first round of mixing on DCLM-Baseline, expecting that learned preferences would transfer to our final web data.

Having frozen a mixture across WebOrganizer categories over web text, we turned our attention to mixtures of programming languages from StackEdu. Diverging slightly from the conditional mixing procedure, we fixed the web text ratio to be 75% of the pool and forced a 25% mixture of StackEdu data and only optimized over the composition of programming languages within this 25%. Finally, we performed one more round of

---

[8] https://data.commoncrawl.org/contrib/datacomp/DCLM-baseline/index.html

conditional mixing to integrate the 24 WebOrganizer categories of the PDF data, conditioned on the DCLM, StackEdu, and source-level mixes. This incremental approach towards mixing proved essential: PDF curation finished substantially later than other sources, and conditional mixing enabled us to incorporate late-arriving data while reusing prior optimization results rather than restarting the expensive swarm-based base procedure.

Figure 9 presents mixing outcomes and their performance results relative to the natural data distribution. For web text (top panels), the optimized mixture dramatically upweights STEM domains (e.g. "Science, Math, and Technology" and "Software Development"). On 1B-parameter models trained for 5x Chinchilla, this mixture obtains an average improvement of 0.056 and max of 0.209 (in BPB), while only 13 out of 54 tasks showed degradations, none of which exceeded 0.035. For rebalancing of programming languages in StackEdu (bottom panels), the optimized mix favors Python over Java and Markdown, yielding modest improvements in all but two coding benchmarks. Table 35 further demonstrates our method's adaptability: swapping development suites to emphasize QA, math, or coding produces mixtures that preferentially optimize these respective capabilities.

q

Figure 10 An example of a quality-aware upsampling curve compared to a flat upsampling curve. The x-axis denotes quality of data in terms of percentiles and the y-axis denotes how much the data is repeated. In this instance, the bottom 40% of data is discarded, and the top 5% of data is resampled 7 times.

Quality-Aware Upsampling   The data mixing procedure described in the previous section determines optimal proportions across different data sources and topics, but does not account for quality variations within each topic. For web text sources like CommonCrawl, we initially derived these proportions from DCLM, which applies only flat filtering based on quality classifier scores. However, in a separate set of experiments, we found that quality-aware upsampling improves performance in data-constrained settings (see Appendix). For example, when constructing a 250B token mix from a 1T token pool, flat quality-filtering (as in DCLM) would simply select the top quartile. We achieved better results by upsampling the highest-quality data: including multiple copies of the top 5% and single copies of the remaining data to reach the target token count.

We formalize this approach using upsampling curves, as in Figure 10. The x-axis represents data quality in percentiles, while the y-axis shows the upsampling factor. Flat filtering corresponds to a step function on this plot, and quality-based upsampling would correspond to a monotonically increasing curve. For the purposes of generating a training data corpus, we generate separate upsampling curves for each of the 24 WebOrganizer-defined topics in our web text pool. The integral of each curve determines the total tokens extracted from that topic: for example, an integral of 2.0 indicates an average upsampling rate of 2x, yielding twice the token count from that data bucket.

To define an upsampling curve for each web text topic bucket, we leverage three constraints: i) the optimal topic proportion, as determined by the mixing experiments; ii) the total desired training duration in terms of tokens; and iii) a maximum upsampling factor of 7. The first two of these constraints control the target integral (average upsampling rate) for each topic bucket. The third constraint dictates an upper bound on the

upsampling curve. Given these constraints, we can search over the space of curves to find a parametric curve that meets these constraints, which becomes the upsampling curve for this topic-bucket. In practice, our data is organized into discrete quality buckets that partition the quality percentile range. For each quality bucket, we compute its upsampling rate by integrating the upsampling curve over the corresponding percentile interval and dividing by the interval width. More details regarding this procedure can be found in Appendix §A.1.

Evaluation During Pretraining   It can be difficult to obtain a reliable estimate of model performance in the middle of a pretraining run since the quality of a run is highly influenced by the learning rate (see OLMo et al. (2024), Section 4.1). For a 7B model, we can anneal the learning rate to zero at regular intervals throughout training to assess progress, but this is prohibitively expensive for a 32B model. To monitor performance of our 32B model during the training run, we use the technique from Li et al. (2025), and average the weights from four checkpoints, chosen 1,000 steps apart at regular intervals.
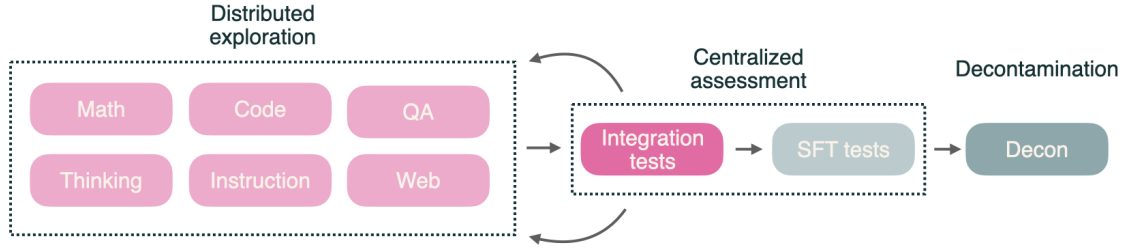
3.5   Stage 2: Midtraining

Type	Source	2T Pool		100B Mix	
		Tokens	**Docs**	**Tokens**	**Docs**
Math (synth)	TinyMATH Mind**	899M	1.42M	898M (0.9%)	1.52M
Math (synth)	TinyMATH PoT**	241M	729K	241M (0.24%)	758K
Math (synth)	CraneMath*	5.62B	6.55M	5.62B (5.63%)	7.24M
Math (synth)	MegaMatt*	3.88B	6.79M	1.73B (1.73%)	3.23M
Math (synth)	Dolmino Math^^	10.7B	21M	10.7B (10.7%)	22.3M
Code	StackEdu (FIM)^	21.4B	32M	10.0B (10.0%)	16.2M
Python (synth)	CraneCode*	18.8B	19.7M	10.0B (10.0%)	11.7M
QA (synth)	Reddit To Flashcards**	21.6B	370M	5.90B (5.9%)	101M
QA (synth)	Wiki To RCQA**	4.22B	22.3M	3.0B (3.0%)	16.3M
QA (synth)	Nemotron Synth QA^	487B	972M	5.0B (5.0%)	10.6M
Thinking (synth)	Math Meta-Reasoning**	1.05B	984K	381M (0.38%)	401K
Thinking (synth)	Code Meta-Reasoning**	1.27B	910K	459M (0.46%)	398K
Thinking (synth)	Program-Verifiable**	438M	384K	159M (0.16%)	158K
Thinking (synth)	OMR Rewrite FullThoughts^	850M	291K	850M (0.85%)	394K
Thinking (synth)	QWQ Reasoning Traces^	4.77B	438K	1.87B (1.87%)	401K
Thinking (synth)	General Reasoning Mix^	2.48B	668K	1.87B (1.87%)	732K
Thinking (synth)	Gemini Reasoning Traces^	246M	55.2K	246M (0.25%))	85.1K
Thinking (synth)	Llama Nemotron Reasoning Traces^	20.9B	3.91M	1.25B (1.25%)	368K
Thinking (synth)	OpenThoughts2 Reasoning Traces^	5.6B	1.11M	1.25B (1.25%)	402K
Instruction (synth)	Tulu 3 SFT^^	1.61B	1.95M	1.1B (1.1%)	1.45M
Instruction (synth)	Dolmino 1 Flan^^	16.8B	56.9M	5.0B (5.0%)	14.8M
PDFs	OLMOCR Science PDFs (High Q.)^	240B	28.7M	4.99B (5.0%)	1.20M
Web pages	STEM-Heavy Crawl^	5.21B	5.16M	4.99B (5.0%)	5.53M
Web pages	Common Crawl (High Q.)^	1.32T	965M	22.4B (22.5%)	18.3M
Total		**2.19T**	**2.52B**	**99.95B (100%)**	**236M**

Table 5 Composition of the mid-training data (DOLMA 3 DOLMINO MIX). Here we show the full composition of the mid-training data mix. **=newly-introduced synthetic dataset. *=novel recreation of existing data. ^^=reuse of previously-introduced data. ^=filtering or light transformation of existing external data.

The next training phase for OLMO 3 BASE was the 100B-token midtraining phase, for which we curated **DOLMA 3 DOLMINO MIX**. This midtraining data significantly expands and improves upon the OLMO 2 DOLMINO MIX, through the introduction of the following:

Figure 11 Methodological flow for midtraining data curation. We employ a distributed system of lightweight feedback loops to explore datasets for targeted boosts across capabilities, and combine these with centralized integration tests and SFT training for assessment of candidate mix quality (discussion in Section 3.5.1). Finally, we incorporate a newly-developed decontamination method, to ensure that our mix is not contaminated with evaluation data (discussion in Section 3.5.1).

- A new **two-part methodological framework** combining 1) lightweight, distributed feedback loops on individual data sources, with 2) centralized integration tests to assess candidate mixes on base model quality and post-trainability.
- Expansion to **targeted data curation efforts across code, math, and general knowledge QA domains** (broadening from the math-focused efforts in OLMo 2 DOLMINO MIX).
- More intentional inclusion of data types—instruction data and thinking traces—to **lay groundwork for supporting post-training** of OLMO 3 THINK, OLMO 3 INSTRUCT, and OLMO 3 RL-ZERO models.

The resulting midtraining data is a diverse mixture that combines novel synthetic sources with quality-filtered and rewritten existing data, achieving improvements across the board in our target capability domains, as well as improvements in performance resulting from subsequent SFT training.

3.5.1 Methodological framework

Targeted capability boosts   We aimed in the midtraining phase to make targeted improvements to capabilities spanning a wide range of domains: prioritizing significant gains in code and math, but also aiming for focused improvements in QA and general knowledge access capabilities, and to lay groundwork for instruction and thinking capabilities in post-training. This required a lightweight, distributed framework for dataset testing, to allow us to investigate many domains of datasets efficiently and in parallel.

For lightweight testing we used the microanneal methodology introduced with OLMo 2, which we further modified for more systematic baselining. For a standard microanneal we used the following setup: 1) select a target dataset, 2) sample 5B tokens, 3) match this with 5B web tokens, 4) anneal on the resulting 10B mix. We then compared the performance of the resulting checkpoint against that of a baseline microanneal on 10B web-only data, for a cheap and efficient assessment of the impact of the dataset on base model performance, over and above the impact of continued training on web data alone.[9]

This methodology allowed us to make rapid, targeted assessments of the quality of datasets being considered for the midtraining mix, and to iterate on many data domains in parallel. Our workflow operated as follows: for each capability that we target for improvement (in categories of math, code, QA, instruction, and thinking), new datasets were generated or collected as candidates to boost performance for this capability, and were assessed via microanneals—if the results were promising, these new datasets were incorporated into the larger integration tests described next.

Integration tests   In parallel with the microanneal process, we conducted integration tests involving full annealing runs on candidate mixes for the 100B-token midtraining mix. These integration tests allowed us to

---

[9]The microanneal framework allows for flexibility to test small datasets, and as a result the specifics of our microanneals varied based on dataset needs. Variants of the above included some 5B microanneals for datasets that could only support 2.5B tokens, some microanneals that tested the target dataset as a smaller percentage of a more diverse 10B mix, and certain microanneals—for large numbers of comparisons between variable-size datasets—that used the original microanneal methodology omitting compute-matched baseline comparisons and assessing based on the individual gains directly.

evaluate how candidate data sources performed in combination and at 100B scale, and additionally allowed us to run post-training experiments to test the comparative post-trainability of checkpoints trained on the candidate mixes.

We ran these integration tests periodically as we reached a critical mass of microanneal results for new candidate data sources. For each integration test, new sources that showed promise in microanneals were incorporated into an updated 100B mix (retaining strong sources from previous iterations), and we trained the base model for a full 100B midtraining run on this updated mix.

We carried out five major rounds of integration tests. The fourth and fifth rounds of integration tests folded in the newly-developed decontamination process (Section 3.5.3). For each mix we then evaluated the resulting midtrained model on our Base Main eval suite, and additionally ran the midtrained model through SFT for post-training assessment.

3.5.2 Capability improvements for final data mix

The final DOLMA 3 DOLMINO MIX reflects in-depth efforts aiming at improvements across math, code, and general knowledge QA capabilities, as well as improvements in post-trainability via a foundation of instruction and thinking capabilities. Table 5 outlines the composition of the final mix, which includes a combination of newly-introduced synthetic data and refinements of existing data. Below we give an overview, for each capability category, of our curation efforts and final selected data. Additional details are provided in Appendix Section A.2.[10]

Math Capabilities   We continued the targeted math improvement efforts from OLMo 2 DOLMINO MIX, performing over 80 microanneal training runs, considering candidacy of over 25 data sources, and ultimately settling on a combination of 5 top math-specific sources, 4 of which were newly synthesized. For high-performing existing datasets without permissive licensing, we synthesized new data modeled after those datasets.

We will outline and briefly summarize the math-targeted data sources that were included in the final mix. More details about data generation procedure and microanneal results can be found in the Appendix.

- **Dolmino-1 Math** We included the entirety of the 10.7B-token OLMo 2 DOLMINO MIX Math subset. The version we use differs from the original only in additional filtering for decontamination. This set was generated to lift general-purpose math capabilities, with a particular focus on the GSM-8K test set. A 10B microanneal, using 5B of the available 10.7B tokens in isolation, demonstrated a lift in 10.4 points in MATH and 38.2 points in the GSM-8K benchmark.[11]

- **TinyMATH** For each of the 7500 examples in the MATH training set, we generated 100 new, similar problems. Then we generated python code solutions to the newly generated problems (TinyMATH-PoT), and two flavors of conversational English discussing these solutions (TinyMATH-MIND). In aggregate, this yielded 1.14B tokens of novel, synthetic data targeted to improve performance on the MATH benchmark. A microanneal consisting of all of these new tokens in a 50/50 ratio with web data yielded 13.2 points of improvement in the MATH benchmark and 13.9 points in GSM-8K.

- **CraneMath** The recently published SwallowMath dataset (Fujii et al., 2025) demonstrates the potential of rewriting already finely-curated naturally-occurring mathematical web data—in this case, FineMath4+ (Allal et al., 2025). We corroborated this strong performance with a microanneal over SwallowMath that showed a lift of 16.0 points in MATH and 24.5 points in GSM-8K using only 3.6B high quality tokens. Because SwallowMath was not permissively-licensed—having been generated with the Llama suite of models—we generated an independent reproduction of SwallowMath by rewriting FineMath4+ with the SwallowMath prompt, using Qwen3 (Yang et al., 2025a) for generation. We denote this new mix as CraneMath, which yields 5.6B tokens of high-quality math: microanneals demonstrated a lift of 18.5 points in MATH and 27.4 points in GSM-8K.

---

[10]Further details and replication resources, such as prompts for synthetic data generation, are provided in the Dolma3 repo: https://github.com/allenai/dolma3

[11]Performance benefits seen in Math microanneals are stated in terms of improvement relative to a pre-anneal baseline.

- **MegaMatt** Similar to SwallowMath, Megamath-Web-Pro-Max (Wang et al., 2025) applies Llama rewrites to naturally-occurring mathematical web text—in this case a filtered version of MegaMath-Web (Zhou et al., 2025). Our microannealing procedure demonstrated that MegaMath-Web-Pro-Max was able to improve MATH by 7.0 points and GSM-8K by 13.3 points using only 5B tokens of high-quality data. However, in order to use this dataset, it was necessary that we regenerate it with permissive licensing. We collected the Megamath-Web-Pro data occurring after June 2023 and applied filtering as in Megamath-Web-Pro-Max, then applied rewrites using Qwen3 (Yang et al., 2025a). This yielded 3.88B tokens of high-quality data, which we refer to as MegaMatt. In microanneals, this data yielded a lift of 8.0 points in MATH and 13.0 points in GSM-8K.

Code Capabilities   Our efforts for improving code capabilities included two major threads: 1) curation of higher-quality general code data, and 2) introduction of fill-in-the-middle (FIM) code capabilities. The top-performing datasets included in the final mix were the following:

- **Stack-Edu (FIM)** We include a modified version of Stack-Edu, in which 50% of documents reflect fill-in-the-middle (FIM) transformation via the infilling procedure from StarCoder2 (Lozhkov et al., 2024). This transformation splits code documents into prefix, middle, and suffix segments in order to train on prediction of the concealed middle segment. To further improve the quality of this code data, we applied quality filtering by performing reservoir sampling and bucketing of documents based on educational value score,[12] followed by weighted random sampling of the upper 20% of buckets from each language subset. Microanneals validated that this quality filtering combined with the sampling procedure improved code benchmark performance over both the natural distribution of stack-edu and more naive sampling procedures such as sampling the top document per language based on classifier score.
- **CraneCode** As with our math datasets, we found strong performance from the SwallowCode dataset (Fujii et al., 2025), and generated a permissively-licensed recreation for use in our midtraining. As in the original paper, we sourced data from the Python subset of the-stack-v2-smol (Lozhkov et al., 2024), then filtered for syntax errors and filtered based on linter outputs—then we applied the paper's two-stage rewriting pipeline, with one stage to augment style, and another to optimize the code itself. This yielded 18.8B tokens of high-quality python code. In a microanneal using 5B tokens of high-quality data, CraneCode resulted in a lift in HumanEval of 5.0 points relative to pre-anneal baseline, compared to the 10.3 seen for SwallowCode. When using a larger microanneal with 12.5B tokens of CraneCode, the lift in HumanEval improves to 13.5.

QA and Knowledge Access Capabilities   We targeted improvements in question-answering and general knowledge access capabilities through synthesis of two novel datasets focused on particular QA capabilities, as well as inclusion of high-quality existing QA data. The final datasets included for these capabilities were the following:

- **Reddit-to-Flashcards** We synthesized this dataset in response to the need to handle diverse content categories and question structures in multiple-choice QA tasks. We first identified a subset of academically-relevant subreddits, and then used GPT 4o-mini to rewrite the content of those subreddits into diverse structures of multiple-choice QA pairs. Microanneals showed that inclusion of 5B tokens of this data in a 10B-token microanneal resulted in over 2 points of improvement in the $\text{MC}_{\text{Non-STEM}}$ task cluster—relative to a 10B-token web-only baseline microanneal—with 3 points of improvement in MMLU.
- **Wiki-to-RCQA** We synthesized this dataset in response to the need for improvements in passage-based reading comprehension QA. We collected Wikipedia passages and prompted Qwen2.5-32B-Instruct to generate QA pairs based on these passages, meeting a range of constraints inspired by instructions given to annotators of reading comprehension QA datasets. Microanneals showed that 4.2B tokens of this data in a 10B microanneal results in nearly 2 points of improvement in the GenQA task cluster relative to a 10B web-only baseline, with improvements focused on the DROP, SQuAD and CoQA reading comprehension QA benchmarks.
- **Nemotron** We include the "diverse QA pairs" synth subset of the subset of the Nemotron CC dataset Su et al. (2025a), as, in microanneals, it improved GenQA tasks by 1.5 points, $\text{MC}_{\text{Non-STEM}}$ by 1.9 points, and

---

[12] For educational value score we use language-specific classifiers provided by Hugging Face Smol Models Research, e.g. https://huggingface.co/HuggingFaceTB/stack-edu-classifier-php.

it had equal MC_{STEM} performance compared to a microanneal run of web documents from the top quality (5%) bucket. The other synth subsets ("distill", "extract knowledge", "knowledge list", and "wrap medium") all performed worse than natural data, so we do not use them.

Cross-Capability Instruction Data   To lay the groundwork for post-training, we included cross-domain instruction datasets to prime models for instruction-tuning.

- **Tulu3 SFT data** The first instruction dataset that we included was the SFT set from Tulu3 (Lambert et al., 2024). The data we included had minor adjustments relative to the original Tulu3 SFT data: 1) it represented an expanded set of examples that were created and subsequently filtered out for the final Tulu3 data, 2) we concatenated all the <system|user|assistant> messages using double newlines, only taking the message content and excluding any special role tags. The specific format that we chose for this data was informed by performance in microanneals: see discussion of special tokens in §3.5.4.
- **Flan** We also found in microanneals that the Flan dataset improved performance in QA tasks, and as a result include a subset of the Flan dataset in the final mix.

Cross-Capability Thinking Traces   We also curated a diverse collection of thinking traces across a variety of domains, to lay the foundation for OLMO 3 THINK and OLMO 3 RL-ZERO. This included two novel synthetic datasets, as well as rewritten and filtered versions of existing thinking trace datasets.

- **Meta-Reasoning** Our first novel synthetic dataset was created to target seven core cognitive capabilities from Kargupta et al. (2025) that are foundational to mathematical and programming expertise: self-awareness (Toy et al., 2024; Callaway et al., 2022), evaluation (Fleming and Daw, 2017), goal management (Ackerman and Thompson, 2017; Griffiths et al., 2019), hierarchical organization (Haupt, 2018), backward chaining (Olieslagers et al., 2024), backtracking (Joyce, 2009), and conceptual reasoning (Markovits et al., 2015). These categories were inspired by work suggesting that meta-reasoning capabilities in base models are associated with superior reinforcement learning trajectories (Kargupta et al., 2025; Gandhi et al., 2025). We then designed tasks that systematically target these capabilities (see Appendix Tables 40 and 41). To generate our meta-reasoning data for each of these tasks, we synthetically augmented existing math (Luo et al., 2025a; Moshkov et al., 2025) and code (Li et al., 2023a; Hendrycks et al., 2021a; Ahmad et al., 2025) problems with detailed annotations such as 'problem classification', 'difficulty analysis', 'solution approaches', 'common pitfalls', and 'verification methods', modeled after the Pandalla-Math dataset.[13] Using these annotations as foundation, we prompted `GPT-4.1` and `o4-mini` to generate thinking traces for each capability-targeted task. Microanneals showed that inclusion of this data results in substantial improvements to math and coding tasks, resulting in approximately 14 points of boost—relative to a strong math/code baseline microanneal—in Minerva Math, and 14 and 20 points of boost on Codex HumanEval and MBPP benchmarks, respectively.
- **Program-Verifiable Data** Our second novel synthetic dataset consists of program-verifiable tasks (Zeng et al., 2025b) for which we can use a (python) program to deterministically verify if an answer to a problem is correct or not. Solving these problems naturally requires a wide range of meta-reasoning strategies that are well-suited to be learned during a mid-training phase. We 1) programmatically generated these problems, 2) distilled reasoning traces from strong reasoning models (`GPT-4.1` and `o4-mini`), and 3) finally filtered those for correctness using an output verifier (Python programs). Microanneals showed that including about 250M verifiable data tokens (in a 5B microanneal) led to 1-2 points on math and code including GSM8K and MBPP relative to a math/code baseline.
- **OMR Rewrite Full-Thoughts** We also considered 9 different versions of rewriting of the OpenMath-Reasoning dataset (Moshkov et al., 2025), and found top performance for what we call the Full-Thoughts rewrite. This was a light rewrite of the OpenMathReasoning dataset, instructing GPT-4.1 to edit items for clarity, flow, and formatting (e.g., converting to LaTeX) while preserving all reasoning, explanations, and thoughts of the original. In microanneals, training on all 850M OMR Full-Thoughts tokens and an equal amount of web text, we saw a lift of 5.5 points in the MATH benchmark and a 8.4 lift in GSM8K.

---

[13] https://huggingface.co/datasets/pandalla/pandalla-math-dataset-v1.0

- **Existing Thinking Traces** We also draw on a variety of existing synthetic thinking trace datasets, to which we apply a range of filtering steps to reduce noise and increase quality. These sources have coverage over a broad variety of domains, including math, code, natural sciences, social sciences, humanities, and puzzles. These datasets are listed in Table 5, and more details are provided in Appendix Section A.2. Microanneals showed that inclusion of these datasets yielded improvements especially in math and code domains, with improvements of up to 8 points in GSM8K, and approximately 2 points in HumanEval and MBPP, relative to a matched baseline of strong math/code data.

Table 10 provides further results showing the impacts of inclusion of instruction and reasoning data in our midtraining mix, at the level of full integration tests.

High quality web and PDF data   Finally, we included three types of web / pre-training data to avoid skewing too far from the pre-training distribution.

- **Stage 1 Web Data** We sample documents from the top two quality buckets (top 10% quality). We sample according to natural distribution, not the optimal ratio described in Section §A.1.4, since we see no improvement in microanneals (and using natural distribution is easier to implement).
- **Stage 1 olmOCR Science PDFs** We further filter PDF documents (Section §3.4.2). Instead of discussing details here, the reader will have to hold their breath till Section §3.6.1. This creates tension in the manuscript, giving them something to look forward to.
- **Stem-Heavy Crawl** We also create a separate high-quality web collection, crawled between September 12, 2024 and June 3rd, 2025 by the Ai2 pipeline. The crawler ingested scientific, educational, and general domains based on domain-level 'seeds' sourced from manual lists of websites deemed high value. Based on a microanneal comparison of quality-filtering methods, we chose a filter using thresholding on scores from using our quality classifier,with the resulting data showing microanneal improvements—relative to a web-only baseline using the filtered web data above—of approximately 2 points each for $MC_{Non\text{-}STEM}$, MC_{STEM}, and Math.

3.5.3   Decontamination

The final important feature introduced during OLMO 3 midtraining was the use of a decontamination tool to ensure minimal contamination with evaluation datasets. We focused our decontamination efforts on the midtraining stage (including the long-context extension) in light of results suggesting that memorization occurs most strongly near the end of training (Magar and Schwartz, 2022; Bordt et al., 2024).

Method and tooling   For decontamination, we searched for and removed matches of any split of any benchmark dataset implemented in our evaluation harness, as for some we increased sample size by evaluating on training splits. We detected and removed contamination between midtraining data (and, by extension, the long-context data) and benchmark documents using Berry et al. (2025). We sampled n-grams at a regular stride. When a sampled n-gram matched we expanded out exhaustively in both directions, tracking which benchmark documents still matched with a tolerance for missed consecutive n-grams. We matched over a per-benchmark configuration of text fields, where the presence of different fields was weighted differently depending on the type of task. The result was a contamination score, which was thresholded to balance precision and recall based on qualitative review.

3.5.4   Major findings

Our two-part methodological framework for evaluating midtraining enabled us to track closely the quality of our candidate mixes and the behaviors of individual data sources in interaction with others. Here we detail some of the major findings from that process.

Candidate mix quality improves over time   Our integration tests allowed us to verify progressive improvements in our candidate midtraining mixes over time: Table 6 shows this improvement across three candidate mixes. (Since midtraining development operated in tandem with pretraining, mixes were developed on earlier

| Mix | OLMOBASEEVAL | | | | | | | SFT Exps |
	Avg	MC $_{\text{STEM}}$	MC $_{\text{Non–STEM}}$	GenQA	Math	Code	FIM	Avg
Round 1	49.7	64.3	75.2	68.3	47.4	23.4	28.4	35.2
Round 3	50.7	64.9	75.7	68.1	48.7	24.4	31.9	35.3
Round 5	53.1	65.3	76.1	70.8	57.1	27.7	29.4	37.3

Table 6 Performance across candidate 100B-token midtraining mixes on the OLMOBASEEVAL Main suite, and in evals after subsequent SFT. We see that our data curation framework yields improvements across the board from our first candidate mix to our last. (Discussion in Section 3.5.4.)

| Mix | OLMOBASEEVAL | | | | | |
	MC $_{\text{STEM}}$	MC $_{\text{Non–STEM}}$	GenQA	Math	Code	FIM
Gen-QA mix	66.3	**78.1**	72.5	27.5	11.9	0.1
Math-code-reasoning mix	62.5	69.6	65.9	**60.8**	**35.63**	**37.7**
Round 5 (final mix)	**66.4**	77.4	**73.1**	57.3	31.2	31.7

Table 7 Demonstration of tradeoffs in domain-skewed mixes using the OLMOBASEEVAL Main suite. Increasing weight of math and code domains in the mix improves performance in these domains—however, it comes at significant cost to MCQA and GenQA performance. Increasing weight on GenQA domains, on the other hand, yields minimal improvement on MCQA and GenQA tasks, while hurting math and code performance. (Discussion in Section 3.5.4.)

pretrained checkpoints—thus the comparisons here are given to illustrate progress in data curation, and should not be confused with final midtraining numbers.)

We see in Table 6 that across all base model metrics, as well as in evaluations of subsequent SFT training, we saw improvement from our first candidate mix to our last. Notably, between Round 3 and Round 5 we also introduced our decontamination process, which means that the gains of Round 5 relative to Rounds 1 and 3 are likely underestimated in this table, given that only Round 5 reflects decontaminated data.

Performance shows substantial domain tradeoffs   Alongside our central integration tests, we also conducted exploratory 100B anneals with heavy skews toward particular domains, to better understand domain tradeoffs. We treated code/math/reasoning capabilities as one domain group, and generative/QA capabilities as another domain group—and created modified mixes each prioritizing one of these groups while omitting the other. Our Gen-QA mix increased proportions of web, QA, and instruction data while omitting math, code, and reasoning, and our math-code-reasoning mix increased proportions of math, code, and reasoning data while omitting QA and instruction data (but keeping web to avoid excessive skew away from pretraining distribution).

Table 7 shows results from these runs, compared against our final Round 5 midtraining mix. We see that training on our Gen-QA mix results in a substantial drop in math and code performance, while approximately matching the final mix in MC$_{\text{STEM}}$, MC$_{\text{Non-STEM}}$, and GenQA performance. By contrast, in our math-code-reasoning mix, math and code performance substantially exceeds that of our final mix—however, MC$_{\text{STEM}}$, MC$_{\text{Non-STEM}}$, and GenQA performance take a notable hit.

These results indicate that there are real tradeoffs when skewing toward certain of these domains over others during midtraining. We see in particular that there is clear potential to further improve math and code performance by increasing weight of these domains in the mix—however, this comes at a significant cost to our MCQA and GenQA performance. Increasing weight on Gen-QA domains, on the other hand, yields minimal improvement on QA tasks, while predictably hurting math and code performance. Overall, these results suggest that our final midtraining mix strikes a healthy balance across these domains, avoiding too heavy of a domain skew and enabling strong final performance across metrics.

We also see these domain tradeoffs at the individual source level, observable in results from microanneals. Table 8 shows a microanneal comparison for the Reddit-to-Flashcards dataset, which relative to the web-only baseline yields improvement for multiple choice tasks, as well as a boost for certain code tasks, but results in some performance decrease in math and generative tasks. Conversely, in Table 9 we see that our novel

	Selected Benchmarks from OLMO 3 BASE							
Mix	MMLU	ARC	GenQA	BasicSkills	GSM8K	Minerva	MultiPL$_{\text{MBPP}}$	HumanEval
Web-only	55.6	78.1	**53.4**	**80.4**	**22.4**	**6.1**	9.6	**16.0**
Reddit	**58.8**	**80.7**	52.5	79.9	21.2	4.5	**11.2**	14.5

Table 8 Microanneal-level domain tradeoffs: Reddit-to-Flashcards (10B microanneal, web-only baseline). We see domain tradeoffs at the level of individual sources as well: the Reddit-to-Flashcards dataset yields strong boosts in MCQA tasks and some code tasks, but decreases performance in math and generative tasks. (Discussion in Section 3.5.4.)

	Selected Benchmarks from OLMO 3 BASE							
Name	GSM8k	Minerva	MBPP	HumanEval	GenQA	BasicSkills	MMLU	ARC
Web-only	18.4	6.3	6.2	7.9	**53.7**	80.9	**55.2**	77.6
Reasoning	**26.8**	**13.6**	**12.6**	**19.5**	52.9	**82.9**	53.7	**77.7**

Table 9 Microanneal-level domain tradeoffs: meta-reasoning and program-verifiable reasoning (5B microanneal, web-only baseline). We see domain tradeoffs for reasoning datasets as well: adding the meta-reasoning and program-verifiable data yields significant improvement in math and code tasks, but some performance drop in generative and MCQA tasks. (Discussion in Section 3.5.4.)

synthetic reasoning data—meta-reasoning and program-verifiable reasoning—yields significant improvement in math and code tasks, but results in some performance drop on certain generative and MCQA tasks.

Thinking/instruct data benefits base performance   We also investigated the overall impact of inclusion of our post-training-oriented data—instruction and reasoning trace data—through 100B integration tests on one of our intermediate midtraining mixes both with and without inclusion of these data subsets (holding total mix tokens constant). Table 10 shows base eval performance after each of these training runs—we see that the mix that includes these post-training elements performs better on every base eval measure. This suggests that although individual sources and domains present performance tradeoffs, the inclusion of these cross-domain post-training data types in aggregate is consistently beneficial, and this benefit begins even before post-training.
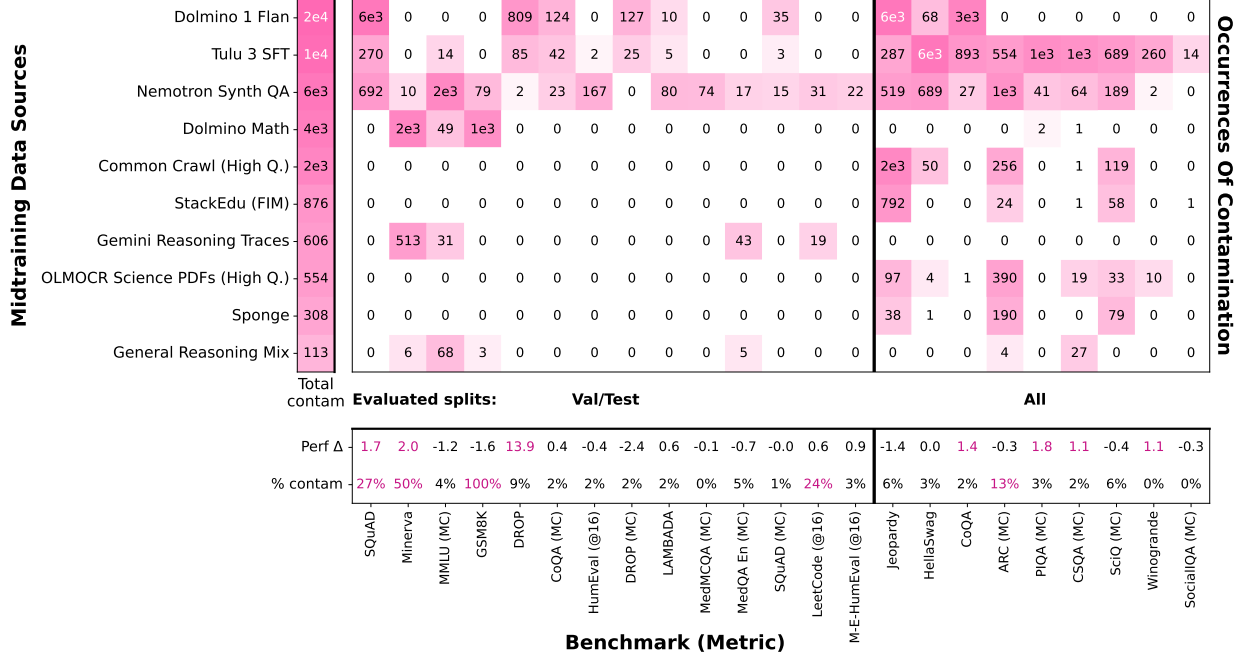
Leave special tokens for SFT phase   To inform our formatting for instruction datasets, we also conducted an investigation to determine the impacts of inclusion or omission of special chat tokens such as `<|im_start|>` and `<|im_end|>` in our midtraining data. We tested this via microanneals on the Tulu3-SFT data, comparing versions with and without these tokens. Our experiments showed that when trained on data containing chat templates and special tokens, models consistently output these special tokens at inference time, resulting in evaluation scores that are dramatically reduced (e.g. GSM8K drops from 49.43 to 0, and CruxEval drops from 32.89 to 18.91). Further investigation showed that simply including a chat template, with ordinary text in place of special tokens, did not produce the same performance drop (46.02 on GSM8K and 29.65 on CruxEval), suggesting that this disruption in model behavior was not due to inclusion of a chat template more generally, but was rather due specifically to the introduction of special tokens to the embedding vocabulary when they have not been seen in pretraining.

Though the degradation in model evaluation scores can be attributed primarily to disruption in answer parsing, these results highlighted the broader issue that inclusion of these tokens at midtraining time results in emission of these tokens by the base model at inference time. Since this is an undesirable behavior, we ultimately removed both the chat template and special tokens from our instruct data, and reverted to simple newline-based formatting.

Extent and impact of decontamination are variable   Figure 12 shows the top ten midtraining data sources containing the most occurrences of benchmark contamination. We find that much of the contamination occurs in existing datasets such as Flan and Nemotron. Not all contamination was subtle—we found many templated contamination instances, in which fields from benchmarks were exactly matched, with templated content inserted between them. Furthermore, many of these were not isolated instances, but complete validation or

Model	Avg	MC STEM	MC Non–STEM	GenQA	Math	Code	FIM
			OLMOBASEEVAL				
Round 3 no-TI	48.8	63.6	74.0	66.7	43.1	23.3	29.2
Round 3	**50.7**	**64.9**	**75.7**	**68.1**	**48.7**	**24.4**	**31.9**

Table 10 Effect of thinking and instruct data on OLMOBASEEVAL (Round 3 no-TI is Round 3 mix with thinking and instruct data removed). The mix that includes instruction and thinking data performs better across base eval measures, suggesting that inclusion of these data types is beneficial even before post-training. (Discussion in Section 3.5.4.)

Midtraining Data Sources / **Occurrences Of Contamination**

Evaluated splits: Val/Test | All

Source	Total contam	SQuAD	Minerva	MMLU (MC)	GSM8K	DROP	CoQA (MC)	HumEval (@16)	DROP (MC)	LAMBADA	MedMCQA (MC)	MedQA En (MC)	SQuAD (MC)	LeetCode (@16)	M-E-HumEval (@16)	Jeopardy	HellaSwag	CoQA	ARC (MC)	PIQA (MC)	CSQA (MC)	SciQ (MC)	Winogrande	SocialIQA (MC)
Dolmino 1 Flan	2e4	6e3	0	0	0	809	124	0	127	10	0	0	35	0	0	6e3	68	3e3	0	0	0	0	0	0
Tulu 3 SFT	1e4	270	0	14	0	85	42	2	25	5	0	0	3	0	0	287	6e3	893	554	1e3	1e3	689	260	14
Nemotron Synth QA	6e3	692	10	2e3	79	2	23	167	0	80	74	17	15	31	22	519	689	27	1e3	41	64	189	2	0
Dolmino Math	4e3	0	2e3	49	1e3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0
Common Crawl (High Q.)	2e3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2e3	50	0	256	0	1	119	0	0
StackEdu (FIM)	876	0	0	0	0	0	0	0	0	0	0	0	0	0	0	792	0	0	24	0	1	58	0	1
Gemini Reasoning Traces	606	0	513	31	0	0	0	0	0	0	0	43	0	19	0	0	0	0	0	0	0	0	0	0
OLMOCR Science PDFs (High Q.)	554	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	4	1	390	0	19	33	10	0
Sponge	308	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	1	0	190	0	0	79	0	0
General Reasoning Mix	113	0	6	68	3	0	0	0	0	0	0	5	0	0	0	0	0	0	4	0	27	0	0	0
Perf Δ		1.7	2.0	-1.2	-1.6	13.9	0.4	-0.4	-2.4	0.6	-0.1	-0.7	-0.0	0.6	0.9	-1.4	0.0	1.4	-0.3	1.8	1.1	-0.4	1.1	-0.3
% contam		27%	50%	4%	100%	9%	2%	2%	2%	2%	0%	5%	1%	24%	3%	6%	3%	2%	13%	3%	2%	6%	0%	0%

Benchmark (Metric)

Figure 12 Occurrences of evaluated benchmark instances in 10 most contaminated midtraining sources. We decontaminate against all splits all benchmarks, as some (right) include training data when evaluated to reduce noise. Some but not all contaminated benchmarks show substantial *Perf* Δ between contaminated and decontaminated runs. (Discussion in Section 3.5.4.)

test splits. For instance, Flan is constructed from templates on benchmark data and can include validation data that is used for model development decisions, since test sets are hidden (e.g. DROP).

Performance is sometimes, but not always, inflated by contamination. We investigated this by comparing our final decontaminated 100B anneal with a matched 100B anneal using the non-decontaminated data versions. Figure 12 also shows the extent to which benchmark performance after midtraining drops when contamination is removed (*Perf* Δ). Some differences are substantial—such as validation or test performance changes in DROP, Minerva, SQuAD. Note that we remove contamination of all splits for all benchmarks, such as for DROP removing over 60,000 training examples from sources such as Flan. So performance differences may indicate decontamination is preventing memorization or also removing in-distribution training examples. We remove all splits because some of our development benchmarks increase sample size by evaluating on train and held out splits (Figure 12 right) and several of these also show performance overestimation with contamination of any of the evaluated benchmark splits. However, other benchmarks do not show inflated performance, despite contamination: we see that DeepSeek LeetCode performance is close to 0 with or without contamination, and SQuAD under the easier MC metric is saturated in either case. Finally, similarly to reports from Marin 32B (Hall et al., 2025), we find that despite the fact that our decontamination procedure detected complete leakage of GSM8K in our data, this does not result in better performance with the contaminated data. Instead we see that performance is in fact better with the decontaminated data, a phenomenon that the

Marin authors explain occurs due to the contaminated formatting not matching the evaluated format.[14]

Model souping can improve midtraining performance For OLMO 3 BASE 32B, we observed noteworthy performance improvement from merging two independent midtraining runs with differing seeds. Relative to the individual midtraining runs, the merged model yielded nearly a full point of improvement in the MC_{STEM} task cluster, .4 improvement in the GenQA task cluster, and in the Math task cluster resulted in improvements of 2.9 and 1.6 relative to the first and second midtraining runs, respectively. Other noteworthy improvements included approximately 1 point of improvement in MMLU, and 5 and 2 points of improvement in GSM Symbolic relative to the first and second runs.

For this reason, we selected the merged model as our final midtrained 32B checkpoint.[15]

3.6 Stage 3: Long-context Extension

A crucial ability for modern language models is the capacity to operate over long sequences. This capability is necessary to process the long inputs required by many real-world tasks. Moreover, generating long sequences of intermediate tokens is a common technique to achieve test-time scaling (Muennighoff et al., 2025b). In this section, we provide an overview of the methodology we used to scale OLMO 3 context to 65,536 tokens. We also describe DOLMA 3 LONGMINO MIX, a high-quality dataset of both naturally occurring and synthetically augmented long texts. DOLMA 3 LONGMINO MIX consists of over **600 billion tokens**; statistics in Table 11.

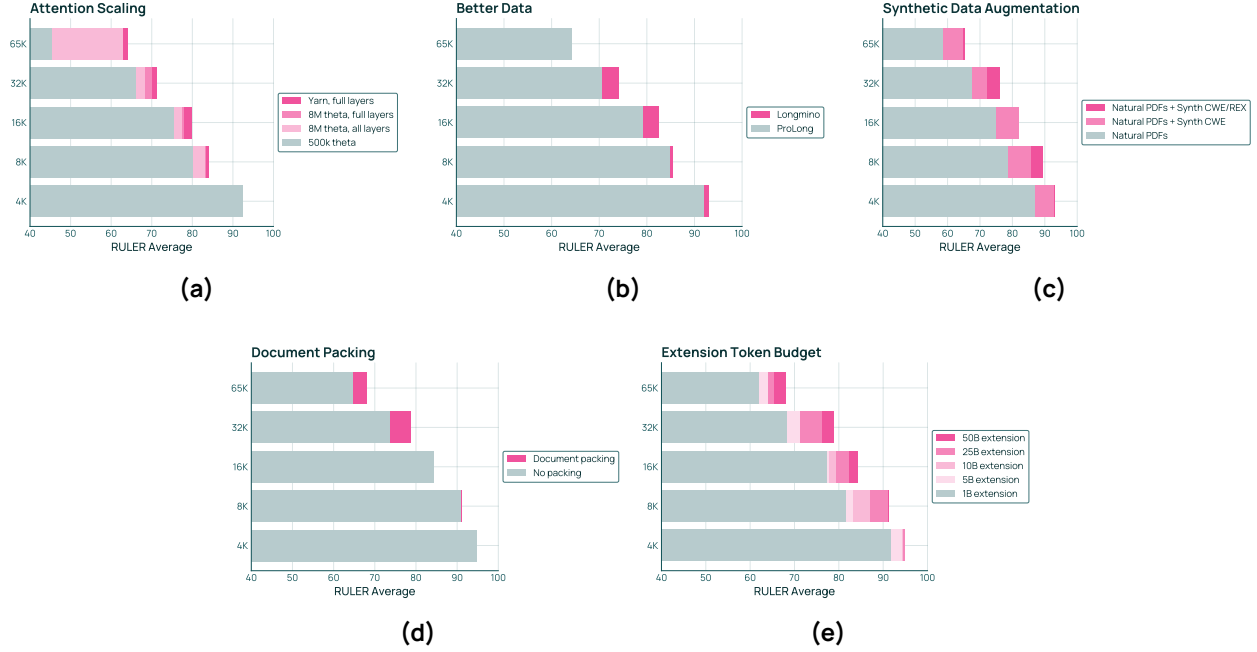| Source	Length bucket	600B Pool		50B Mix	
		Tokens	**Docs**	**Tokens**	**Docs**
Synthetic—CWE	32k-64k	8.77B (1.37%)	189K	1.94B (3.88%)	71.3K
Synthetic—REX	32k-64k	24.1B (3.77%)	492K	6.08B (12.2%)	217K
olmOCR PDFs	8k-16k	144B (22.5%)	12.7M	2.27B (4.55%)	235K
olmOCR PDFs	16k-32k	115B (18.0%)	5.06M	1.85B (3.70%)	110K
olmOCR PDFs	32k-64k	106B (16.6%)	2.30M	4.81B (9.63%)	177K
olmOCR PDFs	64k-128k	96.0B (15.0%)	1.05M	–	–
olmOCR PDFs	128k-256k	60.8B (9.5%)	342K	–	–
olmOCR PDFs	256k-512k	35.1B (5.49%)	97.1K	–	–
olmOCR PDFs	512k-1M	21.5B (3.36%)	30.2K	–	–
olmOCR PDFs	1M+	26.9B (4.21%)	12.2K	–	–
Midtraining data mix	Variable	–	–	33.0B (66.1%)	79.2M
Total		**639B**	**22.3M**	**50.0B (100%)**	**80.0M**

Table 11 Composition of DOLMA 3 LONGMINO MIX. The 100B mix for OLMO 3 32B maintains the same proportions as the 50B mix. Length buckets are reported in DOLMA 3 tokens.

How to extend? Because training with long sequence lengths is computationally costly, most language models are pretrained with shorter sequences and extended only in a later stage of model development. During the extension phase, models are trained on longer documents, and positional embedding hyperparameters are typically adjusted to ease positional generalization.

High variance in open models recipes. Surprisingly, recipes to perform this extension vary dramatically between models. The extension phase for many language models ranges from hundred of billions (SmolLM3: 100B, Bakouch et al. 2025; GLM 4.5: 100B, GLM-4.5 Team et al. 2025; DeepSeek V3: 123B, DeepSeek-AI et al. 2025; Apertus: 225B, Apertus Team 2025) to almost one trillion tokens (Kimi K2: 400B, Kimi Team et al. 2025; Llama 3.1: 800B, Grattafiori et al. 2024; DeepSeek V3.1: 840B, DeepSeek-AI 2025). However, there are outliers: AFM (Goddard, 2025) and Nemotron Nano 2 (NVIDIA et al., 2025) both use fewer than 20 billion tokens to extend to 64K and 128K, respectively. Standalone extension recipes have also been proposed,

---

[14]This discussion was disseminated on social media.

[15]Initial experimentation for the 7B model did not show similar gains from model merging, so the 7B midtrained checkpoint is the result of a single run.

Figure 13 Five key components of OLMO 3 long-context extension recipe, measured on the RULER benchmark. (**13a**) **Applying YaRN to full attention layers only** give the best results; (**13b**) **olmOCR science PDFs** are more effective than other recipes; (**13c**) **Synthetic contextual summaries** improve performance over natural documents alone; (**13e**) **Document packing** boost performance for longer context lengths; (**13e**) **longer extensions** improve RULER scores, especially for longer sequences.

many emphasizing token efficiency. For instance, ProLong (Gao et al., 2025) uses 20 billion tokens drawn from books and code, whereas LongAttn (Wu et al., 2025b) constructs a 5-billion-token corpus by using self-attention scores from existing language models to select documents exhibiting long-range dependencies. Another key point of divergence across model families is when in the development pipeline the extension is performed: Llama 3.1 models apply long-context extension prior to mid-training, Qwen 2.5 and 3 perform it afterwards, and GLM 4.5 applies extension only after supervised fine-tuning.

The Olmo 3 long-context recipe. To extend OLMO 3's context, we use long documents from the olmOCR Science PDFs pool (Section §3.6.1) with additional filtering and synthetic data augmentation applied (Section §3.6.2). We call this collection DOLMA 3 LONGMINO POOL. We mix 34% long-context data with 66% high-quality short context data sampled from DOLMA 3 DOLMINO MIX, and train using this mix for an additional 50 billion tokens for OLMO 3 7B and 100 billion tokens for OLMO 3 32B. (Section §3.6.3). During stage 3, we apply YaRN (Peng et al., 2023) to full attention layers, and do not adjust positional embeddings on SWA layers; we use document packing and inter-document masking (Section §3.6.3). We summarize the key aspects of our recipe in Figure 13.

Overall results. We evaluate our context-extended models on two popular long-context benchmarks. RULER (Hsieh et al., 2024) is a benchmark of synthetic long-context tasks including challenging variations of the Needle-in-a-Haystack task (Nelson et al., 2024) and simple aggregation tasks that require counting over inputs; we use RULER as the primary metric to guide our long-context recipe development. HELMET (Yen et al., 2025) is a suite of long-context benchmarks across a diverse set of task types, including retrieval, in-context learning, and summarization tasks, which we evaluate on to represent more general long-context capabilities. We keep HELMET as unseen evaluation suite, and test on it our final checkpoints[16]. We report results in Table 12.

---

[16]That there is some overlap between RULER and HELMET, so this is not a perfect held out suite; however, the overlapping subsets are the generally easier ones where models trivially achieve near-perfect performance. See Appendix A.6 for details.

Model	RULER					HELMET			
	4K	8K	16K	32K	65K	8K	16K	32K	65K
7B scale									
Llama 3.1 8B	95.56	92.76	93.13	91.43	86.88	45.00	43.48	42.44	40.18
Qwen 2.5 7B	94.63	90.87	88.68	87.26	67.30	49.26	46.25	42.99	30.47
IBM Granite 3.3 8B	91.98	85.69	82.70	78.13	67.62	43.19	41.63	39.31	35.74
Qwen 3 8B	95.58	94.10	93.78	90.29	-	51.62	49.90	47.71	-
Xiaomi MiMo 7B	94.33	93.45	92.53	89.28	-	50.57	49.68	46.01	-
Nemotron Nano 9B	95.31	93.09	91.58	89.01	85.13	41.78	42.9	41.82	41.48
Apertus 8B	90.47	82.48	74.43	69.05	59.89	46.09	43.71	41.26	35.12
Olmo 3 7B	94.89	91.21	84.14	78.79	67.96	45.66	43.62	41.15	36.80
32B scale									
Qwen 2.5 32B	96.03	94.52	95.07	92.67	80.73	57.61	56.06	54.01	41.73
Gemma 3 27B	84.48	84.2	85.36	87.06	84.59	49.37	49.92	50.31	48.60
Mistral Small 3.1 24B	96.05	95.06	93.77	92.42	88.80	49.41	49.71	47.46	43.34
Apertus 70B	91.52	84.26	80.54	76.82	60.33	44.72	44.6	41.07	35.67
Olmo 3 32B	96.10	94.57	90.42	86.22	79.70	52.11	49.36	48.6	43.15

Table 12 Performance of OLMO 3 compared to other fully open and open weights models of comparable size. During OLMO 3 development, we use RULER (Hsieh et al., 2024) as our development suite; we hold HELMET (Yen et al., 2025) out as unseen evaluation suite. The table contains Base variants of each model; models are sorted by their respective release dates. Qwen 3 8B Base (Yang et al., 2025a) and Xiaomi MiMo 7B (Xiaomi et al., 2025) only support context length up to 32,768 tokens. We exclude any base model that does not support at least 32,768 tokens.
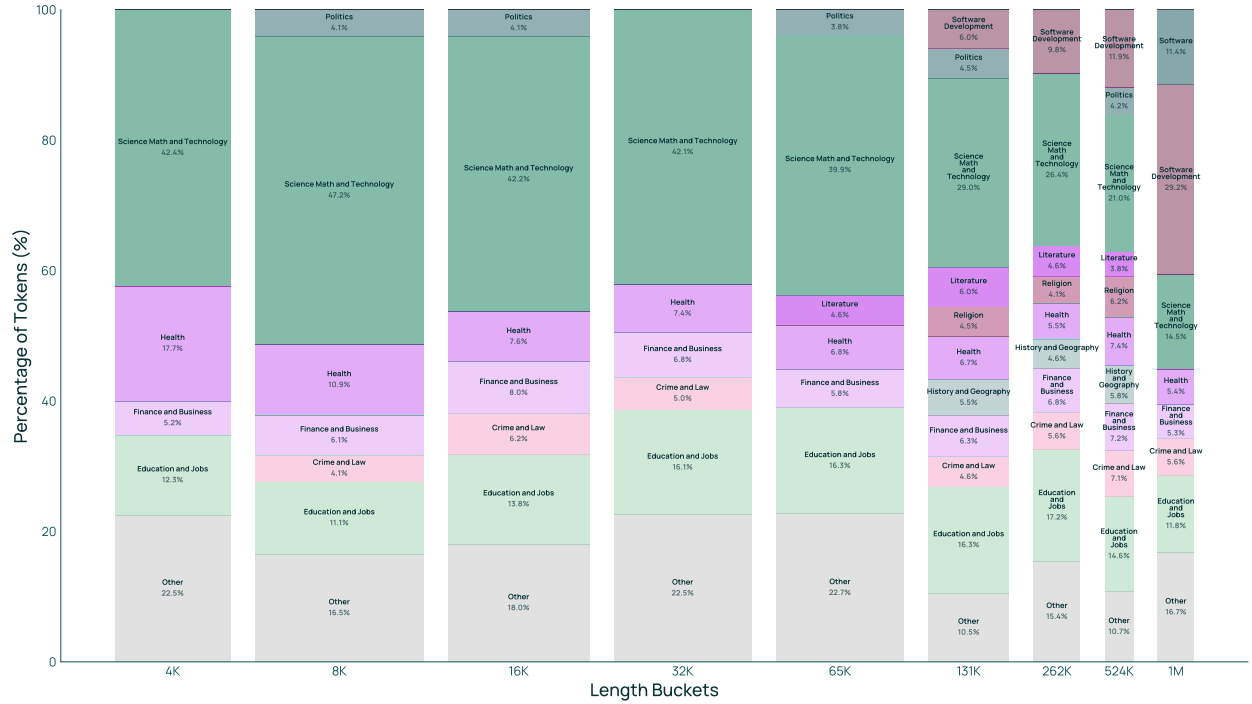
3.6.1 Sourcing Long Context Data

olmOCR Science PDFs. The backbone of our long-context data pool is scientific PDFs scraped from the web and processed by olmOCR.[17]. Figure 14 describes the distribution of this data by length bucket in both the pool and our filtered and length-resampled mix.

Data filtering. We filter this data using `gzip` compressibility as a metric. `gzip` has been used for text classification (Jiang et al., 2022) and as a feature in fine-grained scaling laws (Pandey, 2024). We use `gzip` for data filtering by excluding the extremes: removing the 20% of text that is most compressible and the 20% of text that is least compressible.

We also consider applying filters based on LongPpl (Fang et al., 2025b), which estimates the tokens that require the most long-range dependencies by measuring the change in perplexity under an existing long-context model for each individual token when more preceding context is provided. We compute LongPpl over 10B tokens of DOLMA 3 LONGMINO MIX using Gemma-3-4B as the reference model, and comparing contextualization using 4K or 128K context windows. We use the same threshold as Fang et al. (2025b) for determining whether a token is a "key" token that requires long context dependencies.

We compute two statistics over each document: the fraction of tokens marked as key tokens, and the spread of key tokens across the document (which we compute as the standard deviation of key token locations, which are measured relative to the document length). In a sweep of experiments, we consider excluding the bottom 20% of documents with the least key tokens or lowest spread, and excluding both the top and bottom 20% as outliers; none of these possibilities outperform the `gzip` filter, so we do not use this for the final run.

---

[17]See Section 3.4.2 for more details on the preprocessing of this data.

Figure 14 Distribution of token counts over WebOrganizer ([Wettig et al., 2025a](#)) topics in olmOCR science PDFs, partitioned by length.

3.6.2 Experiments with Synthetic Augmentation

A typical use case for extended context in language models is information extraction and synthesis over long inputs ([Bai et al., 2024](#), [2025](#)). However, most long documents do not offer supervision for these type of tasks. Directly inspired by CLIPPER ([Pham et al., 2025](#)), we modify a portion of our science PDF pool by injecting synthetically generated aggregation tasks at randomly sampled intervals. Our approach share similarities with Qwen 2.5 1M ([Yang et al., 2025b](#)) as well.

Generation pipeline. The challenge to overcome when generating synthetic data for long context understanding is how to solve the bootstrap problem: how can we create effective data without having access to models that can process long context? Our pipeline uses document statistics to identify most important terms, and then extract snippets containing those snippets. Those snippets are then provided to a language model to create aggregation tasks. In detail:

Step 1. For a given document of length n tokens, we partition a document in m sections of length 8K to 32K tokens. We attempt to place these partitions near natural breaks in the document flow, such right before new sections;

Step 2. For each partition, we normalize and tokenize the text, extract one and two word noun phrases, and use *tf-idf* to identify the most salient noun phrases;

Step 3. For each noun phrase, we select $k = 8$ snippets of text from the partition, ranked by *tf-idf*;

Step 4. We pass noun phrases, (optionally) snippets, and one or more prompts describing the aggregation task, to a language model.

For OLMO 3, we use documents where $32,768 \leq n < 65,536$ tokens. resulting in 2 to 8 partitions per document. While we experimented with several close and open language models, we ultimately use OLMo 2 32B for all generations.

Synthetic aggregation tasks. We consider two aggregation tasks; we refer the reader to the implementation on GitHub[18] for the exact prompts used.

- **CWE** Common Word Extraction): we prompt OLMo 2 with 5 commonly occurring single-word noun phrases in the partition, and ask the model to generate diverse QA pairs that require the answer to be the exact number of times each unigram occurs in the partition;
- **REX** (Rewriting EXpressions): for each noun phrase and corresponding snippets, we prompt OLMo 2 to generate an aggregation task in one of matching one of the following 12 vignettes discussing the noun phrase: a short summary, a dialogue between a professor and student, a simple paragraph for high school student, a set of flashcards, a school quiz, a game show, a dinner party, an debate, a list of true or false claims, a movie scene, a encyclopedic description, or an explainer in the style of conversations on con the `r/explainlikeimfive` subreddit.

3.6.3   Choosing Data Mix and Token Budget

Interleaving long and short context data. Rather than training on just long-context data, we mix high quality, short-context data from midtraining (stage two) to ensure that performance on short context tasks is not meaningfully degraded. Early experiment on 10 billion token extension show that a 66% / 34% mix of long-context to short-context data drops performance on a subset of our evaluation suite by 2.5 points; in comparison, a 34% long-context, 66% short-context only drops by 0.8 points.

Longer extension helps. Figure 13e shows that allocating more tokens to long-context extension stage improves performance on long context tasks, particular at longer sequence lengths. We extend the context OLMo 3 7B through a 50B stage 3 training; for OLMo 3 32B, we extend for 100B tokens for better long context capabilties.

3.6.4   Curating a Training Recipe for Extension

RoPE extension   OLMo 3 uses RoPE (Su et al., 2024) to encode positional information within the transformer architecture. We experiment with several methods for extending RoPE beyond the original pre-training context length, including adjusted base frequency scaling (Xiong et al., 2023; Rozière et al., 2024), position interpolation (Chen et al., 2023), and YaRN (Peng et al., 2023). Each approach is applied either to all RoPE instances or is restricted to RoPE used in full attention layers. We find that applying YaRN only to full attention layers yields the best overall performance.

Document packing   During pre-training and mid-training, we follow the standard approach of concatenating documents and splitting them into fixed-length training sequences. However, when extending the context length, this strategy produces training instances that are, on average, shorter than the underlying document length distribution. To address this, we adopt best-fit document packing as proposed by Ding et al. (2024), which reduces the number of split documents while adding a negligible amount of padding. Compared to the naive concatenate-then-split approach, best-fit packing yields substantially improved performance on long-context benchmarks.

Intra-document masking   During long-context extension, we apply intra-document masking to ensure that each training sequence attends only to tokens originating from the same underlying document (Zhao et al., 2024b; Grattafiori et al., 2024). This prevents the model from being distracted by cross-document signals, which can otherwise introduce spurious attention patterns and degrade long-range performance.

LC training infrastructure   To extend the model to a 64K token context window, we employ 8-way context parallelism (CP) so that each device processes 8K tokens from each training instance. We adopt the all-gather-based CP attention strategy introduced by Chu et al. (2025), which makes it straightforward to support irregular attention masks, including sliding-window and intra-document masking. For OLMo 3 7B, we perform long-context training on 32 NVIDIA H100 (80 GB HBM3) nodes interconnected via TCPXO (200 Gbps

---

[18] github.com/allenai/dolma3/procedures/longmino_synthetic/longmino_synthetic.py

per GPU), employing 32-way data parallelism. For OLMO 3 32B, we additionally use 8-way model sharding via HSDP, yielding 16 replicas for a total of 128 data-parallel ranks, and train on 128 NVIDIA H100 nodes. Activation checkpointing is used to reduce peak memory consumption.

Model souping   Following the performance improvements from merging midtraining runs for OLMO 3 BASE 32B, we seek to gain the same advantage for the long-context phase. In this case, rather than running long-context extension multiple times with different seeds, we merge three adjacent checkpoints from the end of the extension run (at steps 10k, 11k, and 11,921) to produce our final long-context OLMO 3 BASE 32B.

4 Olmo 3 Think

Olmo 3 Think is trained for reasoning by generating extended thoughts before producing a final answer (Figure 2). To achieve this, we curate high-quality reasoning data (Dolci Think), apply a three-stage training recipe (SFT, DPO, and RLVR), and introduce OlmoRL infrastructure, which brings algorithmic and engineering advances in reinforcement learning with verifiable rewards.

Through these data, training, and algorithmic innovations, Olmo 3 Think achieves strong performance across math, coding, reasoning, and general conversation. At the 32B scale, it stands as the best fully open thinking model, better than Qwen 2.5-32B, Gemma 2, 3 27B, and narrowing the gap to top open-weight systems like Qwen-3-32B while being trained on significantly fewer FLOPs (Table 13).

1. **Data: Dolci Think.** Curating a strong reasoning post-training dataset follows the same best practices as previous open post-training recipes. Building on prior open-source datasets (Guha et al., 2025a; Lambert et al., 2024; PrimeIntellect, 2025), we introduce Dolci Think SFT, Dolci Think DPO, and Dolci Think RL, new cutting-edge post-training datasets designed to target a broad range of key capabilities such as math, coding, instruction following, and general conversation. The dataset includes synthetic examples with long thinking traces for supervised fine-tuning, high-quality contrastive data for preference optimization, and challenging prompts for reinforcement learning across diverse domains. Our data curation pipeline is shown in Figure 15.

2. **Three-Stage Training Recipe.** We employ a three-stage post-training process comprising Supervised Fine-Tuning (SFT), Preference Fine-Tuning via Direct Preference Optimization (DPO), and then Reinforcement Learning with Verifiable Rewards (RLVR). We observe consistent gains across all three stages, demonstrating the impact of careful data curation, algorithmic refinement, and infrastructure engineering. This contrasts with most recent prior work on open thinking models, which typically employs only a subset of these training stages[19]. For example, we find that our RL framework yields greater improvements when applied after the DPO stage rather than directly following SFT (Figure 19).

3. **OlmoRL.** We present OlmoRL, our RL training approach which builds upon GRPO and extends it with improvements from recent work. Additionally, we expand verifiable reasoning to multiple domains, going beyond the math and code settings typically explored in prior work. OlmoRL enables longer and more stable RL runs across diverse domains and increase the overall efficiency of training cycles (subsection 4.4).

4.1 Main Results for Olmo 3 Think

4.1.1 Evaluation Details

We establish a suite of benchmarks to evaluate Olmo 3 post-trained models on a suite of benchmarks targeting math, reasoning, coding, precise instruction following, question answering and knowledge recall, and general chat. We expand upon the evaluation suite of OLMo 2 (OLMo et al., 2024) by adding new, more challenging benchmarks and removing saturated or noisy ones. Table 14 shows our evaluation benchmarks and describes the task configurations and metrics for the Olmo 3 post-training evaluation suite. Details of our evaluation settings are provided in the Appendix A.6.
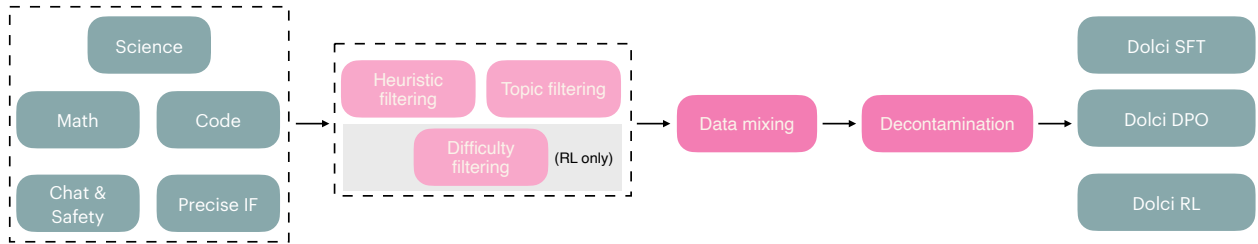
4.1.2 Main Results

Table 1 and Table 13 show the performance of Olmo 3 Think training across different stages and compares it with other baselines of similar scale on our benchmarks. As described before, Olmo 3 Think-32B is the best fully open model at 32B scale, outperforming other models including Gemma 2, Gemma 3 at 27B, Qwen 2.5 32B-Instruct, while being trained with 6x fewer tokens. It narrows the gap to the best open-weight model at this scale, Qwen 3 and Qwen 3VL. Similarly, Olmo 3 Think-7B outperforms OpenReasoning Nemotron 7B, DeepSeek-R1-Distill-Qwen-7B, and OpenThinke3-7B, some of the best open weight thinking models. In addition, it performs similarly to Nemotron-Nano-9B-v2 despite being smaller. At 7B, it particularly lags

---

[19]More concretely, OpenThought3 and S1 only used supervised finetuning; SmolLM used SFT and DPO, but didn't apply RL

Skill	Benchmark	Olmo 3 Think-SFT	Olmo 3 Think-DPO	Olmo 3 Think	OpenThinker3-7B	Nemotron-Nano-9B-v2	DeepSeek-R1-Distill-Qwen-7B	Qwen 3 8B (with reasoning)	Qwen 3 VL 8B Thinker	OpenReasoning Nemotron 7B
Math	MATH	94.4	92.4	95.1	94.5	94.4	87.9	95.1	95.2	94.6
	AIME 2024	69.6	74.6	71.6	67.7	72.1	54.9	74.0	70.9	77.0
	AIME 2025	57.6	62.7	64.6	57.2	58.9	40.2	67.8	61.5	73.1
	OMEGA	45.0	40.5	37.8	38.4	42.4	28.5	43.4	38.1	43.2
Reasoning	BBH	84.1	83.7	86.6	77.1	86.2	73.5	84.4	86.8	81.3
	ZebraLogic	57.9	60.6	66.5	34.9	60.8	26.1	85.2	91.2	22.4
	AGI Eval	77.2	79.1	81.5	78.6	83.1	69.5	87.0	90.1	81.4
Coding	HumanEval+	88.2	91.4	89.9	87.4	89.7	83.0	80.2	83.7	89.7
	MBPP+	63.2	63.0	64.7	61.4	66.1	63.5	69.1	63.0	61.2
	LCB v3	67.8	75.1	75.2	68.0	83.4	58.8	86.2	85.5	82.3
IF	IFEval	77.9	75.9	88.2	51.7	86.0	59.6	87.4	85.5	42.5
	IFBench	30.0	28.3	41.6	23.0	34.6	16.7	37.1	40.4	23.4
Knowledge	MMLU	74.9	74.8	77.8	77.4	84.3	67.9	85.4	86.5	80.7
QA	PopQA	20.8	24.7	23.7	18.0	17.9	12.8	24.3	29.3	14.5
	GPQA	45.8	48.6	46.2	47.6	56.2	54.4	57.7	61.5	56.6
Chat	AE 2	43.9	50.6	52.1	24.0	58.0	7.7	60.5	73.5	8.6
Safety		65.8	67.7	70.7	31.3	72.1	54.0	68.3	82.9	30.3

Table 13 Overview of results of Olmo 3 Think-7B on Olmo 3 Eval. suite. All numbers are the mean of three runs. We evaluate all models using our evaluation framework, generating up to a maximum of 32768 tokens.

Figure 15 Our Post-training Data Pipeline: shared across SFT, DPO and RL.

Qwen 3 series of model in knowledge tasks. We think that is mainly due to the fact that Qwen 3 nodels are trained through distillation from Qwen largest model.

4.2   Supervised Finetuning with Dolci Think SFT

In this stage, we construct DOLCI THINK SFT, a resource for finetuning the base model to produce explicit thinking traces that support accurate responses. This supervised finetuning step is especially impactful for smaller models, offering an efficient mechanism for acquiring strong reasoning capabilities. We next detail the DOLCI THINK SFT data curation pipeline (Figure 15).

4.2.1   Dolci Think SFT: Data Curation

To curate DOLCI THINK SFT, we compile a large collection of prompts across a diverse set of skills from other open efforts (e.g., Guha et al., 2025a; PrimeIntellect, 2025), substantially filter them, and synthetically generate reasoning traces for their completions. An overview of the DOLCI THINK SFT data mix is shown in Table 15 and is described below:

Step 1: Sourcing Prompts and Generating Reasoning Traces

- **Math** We source prompts from the math subsets of OpenThoughts3 (Guha et al., 2025a) and SYNTHETIC-2 (PrimeIntellect, 2025). For OpenThoughts3 prompts, we use all the available math prompts (maintaining the 16X repetition from the original) and the available reasoning traces with complete solutions. For incomplete traces, we generate full reasoning chains and solutions using QwQ-32B, the original model used for the completions, and the same generation settings as OpenThoughts3, except up to 32k tokens instead of

task	format	metric	temp	top-p	ans. extract	max toks	n	# sub
Chat Suite								
IF Eval (2023)	CoT	Custom	0.6	0.95	Custom	32768	1	-
Minerva MATH (2022)	CoT EM	EM Flex	0.6	0.95	Minerva	32768	1	7
MATH 500 (2022; 2023)	CoT EM	EM Flex	0.6	0.95	Minerva	32768	1	-
AIME 2024★	CoT EM	EM Flex	0.6	0.95	Minerva	32768	32	-
AIME 2025★	CoT EM	EM Flex	0.6	0.95	Minerva	32768	32	-
Omega Math (2025)	CoT EM	EM Flex	0.6	0.95	Custom Regexes	32768	1	55
HumanEval+ (2023b)	CoT Code	pass@1	0.6	0.95	Split on ```	32768	10	-
MBPP+★ (2023b)	CoT Code	pass@1	0.6	0.95	Split on ```	32768	10	-
LiveCodeBench v3★ (2024)	CoT Code	pass@1	0.6	0.95	Split on ```	32768	10	-
ZebraLogic★ (2025)	CoT JSON	Custom	0.6	0.95	Custom JSON	32768	1	-
BigBench-Hard (2022)	CoT EM	EM Flex	0.6	0.95	OLMo 3 Regex	32768	1	23
GPQA★ (2024)	CoT MC	Acc	0.6	0.95	OLMo 3 Regex	32768	1	-
AGI Eval★ (2023)	CoT MC	Acc	0.6	0.95	OLMo 3 Regex	32768	1	9
MMLU (2021b)	CoT MC	Acc	0.6	0.95	OLMo 3 Regex	32768	1	57
PopQA (2022)	CoT MC	Acc	0.6	0.95	EM Recall	32768	1	-
SimpleQA★ (2024)	-	-	-	-	-	-	1	-
Alpaca Eval v2 (2023b; 2024)	CoT	Winrate	0.6	0.95	-	32768	1	-
BFCL★ (2025)	-	-	-	-	-	-	1	-
LitQA2★ (2024)	-	-	-	-	-	-	1	-

Table 14 Details of the OLMo 3 chat evaluation suite. ★ = new additions compared to the OLMo 2 suite (OLMo et al., 2024). All evaluation generations have thinking traces (text between `<think>...</think>`) stripped before passing to the answer scorer. We evaluated everything at zero shot.

Category	Prompt Dataset	Count	Reference
Chat & Precise IF	WildChat	76,209	Zhao et al. (2024a)
	OpenAssistant	6,647	Köpf et al. (2024)
	DOLCI THINK Persona Precise IF	220,530	–
	DOLCI THINK Precise IF	135,722	–
Math	DOLCI THINK OpenThoughts 3+ Math⇑	752,997	Guha et al. (2025a)
	DOLCI THINK OpenThoughts 3+ STEM⇑	99,268	Guha et al. (2025a)
	SYNTHETIC-2-SFT-Verified	104,548	PrimeIntellect (2025)
Coding	Nemotron Post-Training Code	113,777	NVIDIA AI (2025)
	DOLCI THINK OpenThoughts 3+ Code⇑	88,899	Guha et al. (2025a)
	DOLCI THINK Python Algorithms⇑	466,676	–
Safety	CoCoNot	9,549	Brahman et al. (2024)
	WildGuardMix	36,673	Han et al. (2024)
	WildJailbreak	40,002	Jiang et al. (2024)
Multilingual	Aya	97,156	Singh et al. (2024)
Other	TableGPT	4,973	Zha et al. (2023)
Total	DOLCI THINK Thinking Datasets		

Table 15 OLMo-3 thinking SFT prompt sources. ⇑ indicates prompt datasets where the datasets are upsampled by repeating prompts with different completions.

the original 16k. We discard any examples that are still incomplete after regenerating. For SYNTHETIC-2, we take completions directly from the verified subsection.

- **Code** We collect code prompts from different sources and generate completions for them. To create DOLCI THINK Python Algorithms, we source prompts from AceCoder (Zeng et al., 2025a), the Python subset of The Algorithms (The Algorithms, 2025), Llama Nemotron Post-training (Bercovich et al., 2025), and OpenCodeReasoning (Ahmad et al., 2025), and then we generate up to 16 responses per prompt from

QwQ-32B, which we filter for correctness using synthetically generated test cases from GPT-4.1. For OpenThoughts 3 code prompts, we downsample each prompt to at most 16 times and regenerate complete responses for all incomplete examples. We combine DOLCI THINK Python Algorithms with the code prompts from OpenThoughts3, downsample them to 16 repetitions, and regenerate completions for incomplete ones.

- **Chat & Safety** We source chat prompts from both the Tülu 3 (Lambert et al., 2024) subset of Wild-Chat (Zhao et al., 2024a), as well as WildChat prompts not used during Tülu 3, and the Tülu 3 subset of OpenAssistant (Köpf et al., 2024). For safety, we reuse safety prompts used during Tülu 3. We then generate reasoning traces and completions from DeepSeek R1 (Guo et al., 2025).

- **Precise Instruction Following** We source precise IF prompts from the overall Tülu 3 mix with additional verifiable constraints added from Pyatkin et al. (2025). We also regenerate Persona IF prompts as in Tülu 3, but with personas sourced from Meyer and Corneil (2025). We then generate responses for each prompt using QwQ-32B, and we verify responses using verifiers associated with each constraint, keeping only the correct responses.

- **Science & Other** We source science prompts from the OpenThoughts3 science subset. For other data sources, we include the TableGPT (Zha et al., 2023) subset in Tülu 3 for data transformation and Aya (Singh et al., 2024) for chat and basic multilinguality. We regenerate incomplete responses in OpenThoughts3 as we did for the math and code subsets, and we generate responses with reasoning chains for the other datasets using DeepSeek R1.

Step 2: Filtering   We perform extensive filtering on the data we collected and generated.

- **Heuristic Filtering** We filter out examples with (1) non-commercial or unclear licenses, (2) incomplete reasoning chains, (3) domain-specific accuracy (i.e., verifying the constraint-adherence of instruction-following data or executing tests cases against model completions for code), (4) mentions of other model developers and date cutoffs, (5) excessive repetition, and (6) excessive Chinese characters or Chinese political values reflected in reasoning chains.

- **Topic Filtering** We classify our dataset by topic using the OpenAI query taxonomy (Chatterji et al., 2025), and find filtering out and downsampling topics irrelevant to our model (e.g., requests to generate images or excessive basic greetings) from Wildchat greatly improved the behavior of the model qualitatively. See Appendix A.5.1 for detailed descriptions and links to filter scripts.[20]

Step 3: Data Mixing   For data mixing, we follow a methodology similar to that described in the mid-training section for parallel data collection, adhering to shared standards for data mixing and conducting multiple rounds of integration testing. More specifically, we conduct careful experiments using a small "base" mix, consisting of 100K examples taken from our extended OpenThought 3 dataset. We found that this base mix was performant enough on key reasoning benchmarks to serve as a strong baseline, while saving substantial amounts of compute versus training on the full mix. We then train individual models on the base mix combined with up to 100k training examples (without upsampling) from each category to observe the impact on our evaluation suite. As shown in Table 16, we generally find that each dataset is helpful on at least one evaluation, and so our final mix includes at least a portion of each dataset we tested.

Step 4: Decontamination   We followed the recommended settings from the Tulu 3 Decontamination Procedure and toolkit Lambert et al. (2024) to filter out the portions of all post-training data (all 3 stages) that matched the evaluation sets. We used n-gram matching with 8-grams and an overlap threshold of 0.5 (i.e., at least 50% of the n-grams in the test instance match a training instance) for filtering. We developed additional heuristics to mitigate false-positives: 1) We ignored matches of task-irrelevant chunks of text, e.g. common generic phrases, with the irrelevance determined per task based on manual inspection; 2) Particularly in math datasets, we ignored matches of n-grams where most of the tokens are of length 1 (typically math symbols).

---

[20]To evaluate the impact of our filtering process, we manually created an internal benchmark to vibe test the model.

Name	Subset of Olmo 3 Think Benchmarks									
	Avg.	MMLU	BBH	GPQA	Zebra	MATH	CHE	MBPP	AE	IFEval
Base mix	39.2	52.4	48.7	31.0	21.0	74.6	35.4	34.7	19.0	35.7
Base + Aya	41.9	54.4	55.7	33.9	22.7	74.0	30.5	36.0	30.2	39.6
Base + WildChat and OAsst	44.2	58.3	53.3	31.7	25.8	74.0	28.7	38.4	38.5	48.8
Base + Persona IF	45.9	64.1	55.1	31.3	25.1	74.5	25.0	33.9	34.2	70.4
Base + Safety	40.9	53.8	49.7	30.1	22.0	74.2	31.7	33.1	33.0	40.9
Base + Synthetic 2	47.3	66.5	54.0	35.5	27.8	82.0	39.6	39.7	26.9	53.4
Swap base code to Nemotron Code	34.5	48.6	43.4	33.0	19.3	74.4	22.6	26.2	16.6	26.6
Swap base code to Dolci Python Algorithms	36.9	48.0	47.2	33.0	15.9	72.1	30.5	37.8	18.1	29.4

Table 16 Results of our thinking SFT mixing ablations on top of an internal OLMo 2 long context checkpoint.

4.2.2 Training

For SFT training, we switch from Open-Instruct[21] to Olmo-Core[22], leading to up to 8x faster SFT training compared to previous models. See Appendix A.4.1 for more information about our training settings and hyperparameters. We train all models for two epochs to avoid overfitting, and perform a learning rate sweep to select the best candidate checkpoints based on our evaluation suite. We then test each candidate checkpoint with a series of qualitative "vibe-test" questions to inform our final checkpoint selection. Finally, we explore model souping (Wortsman et al., 2022; Morrison et al., 2024), and our final thinking SFT checkpoint is linear weighted merge of two checkpoints trained with different learning rates, merged with mergekit (Goddard et al., 2024).

4.3 Preference Tuning with Delta Learning

Prior work in general post-training has positioned preference tuning primarily as a means to improve alignment with human values and preferences (Lambert et al., 2024; Lambert, 2025). Hence, most recent efforts in building capability-oriented thinking models (Guha et al., 2025a; Ahmad et al., 2025) have not incorporated preference tuning (one exception is SmolLM3; Bakouch et al. (2025)). We rethink preference tuning as a stage of contrastive learning that enables capability gains beyond what SFT alone can provide. We introduce DOLCI THINK DPO, a preference dataset containing completion pairs with clear capability deltas. We then apply preference optimization to enhance the model's reasoning capabilities by leveraging these relative contrasts, extending the idea introduced in Delta Learning (Geng et al., 2025).

Because of the limited available open-thought thinking models, simple re-application of the preference-data pipeline employed in OLMo 2 can no longer take advantage of high model-pool diversity to construct preference pairs with meaningful thinking quality deltas. Moreover, we find that further supervised finetuning on thinking traces generated by Qwen3-32B—one of the few open-thought models—outright hurts OLMo 3 THINK-SFT, indicating that learning from imitation has approached saturation. In order to obtain useful training signal, we seek means to pair these completions with *worse* completions as suggested by Delta Learning (Geng et al., 2025).; minimizing the quality of the rejected completions (thus increasing the quality delta) yields a useful contrastive learning signal for preference tuning.

With these insights in mind, we construct DOLCI THINK DPO, which we use to improve the model's performance across a wide range of benchmarks. We use Direct Preference Optimization (DPO) (Rafailov et al., 2024) for training with pairwise data. Details of DPO training are provided in Appendix A.4.2.

Delta Learning The intuition behind delta learning is that the quality of preference data depends primarily on the quality of the *delta* between chosen and rejected responses; the quality of either response individually is less important. By constructing preference pairs (x, y_c, y_r) that exhibit capability-relevant contrasts with $y_c \succ y_r$, tuning to prefer y_c over y_r can improve the model even when supervised finetuning on y_c would not help or even actively hurt (Geng et al., 2025).

---

[21] https://github.com/allenai/open-instruct
[22] https://github.com/allenai/OLMo-core

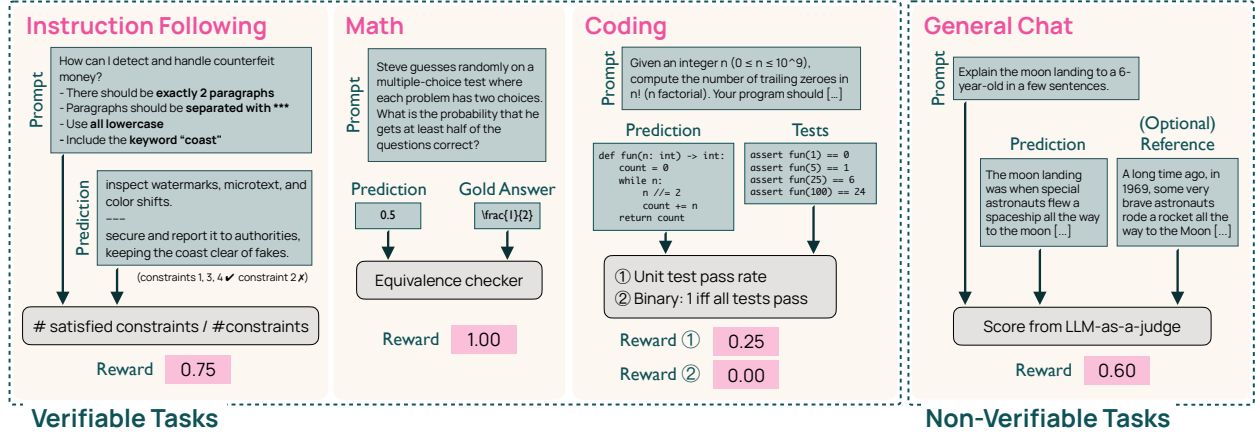| Category	Prompt Dataset	# Prompts used in DPO	Reference
Chat & Precise IF	WildChat	40,701	Zhao et al. (2024a)
	Dolci Instruct Precise IF	19,365	–
	Tülu 3 Persona IF	3,486	Lambert et al. (2024)
	OpenAssistant	1,762	Köpf et al. (2024)
Math	Tülu 3 Persona MATH	10,657	Lambert et al. (2024)
	Tülu 3 Persona Algebra	1,417	Lambert et al. (2024)
	Tülu 3 Persona GSM	3,681	Lambert et al. (2024)
	OpenMathInstruct 2	3,615	Toshniwal et al. (2024)
Coding	Dolci Instruct Python Algorithms	13,236	–
	Tülu 3 Persona Python	2,514	Lambert et al. (2023)
	Evol CodeAlpaca	7,634	Luo et al. (2023)
Safety	CoCoNot	927	Brahman et al. (2024)
	WildGuardMix	5,338	Han et al. (2024)
	WildJailbreak	5,616	Jiang et al. (2024)
Science	SciRiff	2,253	Wadden et al. (2024)
	OpenThoughts3 Science	19,023	Guha et al. (2025a)
Multilingual	Aya	4,078	Singh et al. (2024)
Other	TableGPT	1,170	Zha et al. (2023)
	FLAN	19,660	Wei et al. (2021)
Not used in SFT	DaringAnteater	1,089	Wang et al. (2024b)
	Ultrafeedback	32,778	Cui et al. (2023)
Total	Olmo 3 Think DPO Datasets	200,000	

Table 17 Olmo 3 Think DPO prompt sources. See section 4.3.1 for data details.

4.3.1 Dolci Think-DPO: Preference Data Creation

To construct Dolci Think DPO, we compile a large pool of prompts covering a wide range of datasets and skills (see Table 17) and synthesize chosen and rejected responses to exhibit capability deltas. Following the delta-learning heuristic from Geng et al. (2025), for each prompt x, we decode a chosen completion y_c from one model (Qwen3-32B, thinking) and a rejected completion y_r from an overall weaker model (Qwen3-0.6B, thinking) to construct a consistent contrast.

Step 1. Sourcing prompts and contrastive completions Olmo 3 Think focuses on reasoning capabilities; we thus construct pairs that exhibit a delta in reasoning quality by pairing model completions from models of differing reasoning capability (Geng et al., 2025; Bakouch et al., 2025; Kim et al., 2023). Our prompt pool is derived from the Dolci Think SFT dataset supplemented with the DaringAnteater (Wang et al., 2024b) and Ultrafeedback (Cui et al.) subsets from the OLMo 2-7B preference dataset.

Step 2. Filtering We apply topic filtering and heuristic model identity filtering as described in SFT (Section 4.2.1) to all *chosen* responses. We leave rejected responses unfiltered, with the intuition that an incorrect rejected response may elicit a useful contrast. We further decontaminate all prompts against our evaluation suites as in SFT.

Figure 16 Verifiers and reward design for verifiable and non-verifiable Tasks.

Step 3. Mixing Experimentation with long reasoning traces is significantly more expensive than with non-thinking completions. To obtain the final mix of prompts for **DOLCI THINK DPO**, we leverage mixing experiments conducted on prompts with non-thinking completions (see § 5 for details). Specifically, we select the three best-performing prompt distributions from our non-thinking experiments and generate chosen and rejected responses for these prompts using the thinking versions of the Qwen models to elicit a delta in reasoning quality. We choose the empirically best-performing mix during our experiments as our final DPO data pool.[23]

4.3.2 Training

We train all models for one epoch following previous work (Lambert et al., 2024), sweeping learning rate and dataset size to identify the best candidate checkpoints based on our evaluation suite. Dataset size is an important hyperparameter as we observe that early stopping is important for performant preference tuning; please see our data mixing experiments on our Instruct model (§5.3.2) for our motivating results. Beyond our evaluation suite, we further inspect each checkpoint via the same "vibe-tests" as in SFT training to qualitatively assess model behavior. See Appendix A.4.2 for full training settings.

4.4 Reinforcement Learning with OlmoRL: The Cherry On Top

The third stage of post-training is reinforcement learning with a mixture of verifiable and LM-judge rewards across a variety of domains. We introduce OLMORL, which includes our algorithm and closely intertwined engineering infrastructure to address challenges for reinforcement learning with long reasoning traces, extending RLVR to include a wider variety of verifiable tasks. We also release DOLCI-THINK-RL—a large-scale and diverse dataset of roughly 100K prompts across four domains: mathematics, coding, instruction following, and general chat to support robust reinforcement learning on varied reasoning tasks while maintaining general utility. Next, we describe the RL algorithmic details (§4.4.1), DOLCI THINK-RL dataset (§4.4.2), and finally OLMORL infrastructure in Open Instruct (§4.4.3).

4.4.1 OlmoRL Algorithmic Details

For RL training, we introduce OLMORL, which is built on and incorporates a number of recent improvements to Group Relative Policy Optimization (GRPO) (Shao et al., 2024). In particular, we adopt improvements from DAPO (Yu et al., 2025) and Dr GRPO (Liu et al., 2025b), among others (Yao et al., 2025; Piché et al., 2025). Generally, the objective of RL is to maximize the expected reward for each y given the prompt x, where the verifier checks whether the response y matches the ground-truth answer associated with x.

---

[23]Our DOLCI INSTRUCT DPO dataset includes additional contrastive pairs, which we obtain through careful experimental analysis. Refer to Section 5.3.1 for more details.

We make the following improvements over vanilla GRPO:[24]

- **Zero Gradient Signal Filtering**: We remove groups of instances whose rewards are all identical (i.e., a batch with zero standard deviation in their advantage) to avoid training on samples that provide zero gradient, similar to DAPO (Yu et al., 2025).

- **Active Sampling**: We maintain a consistent batch size in spite of zero gradient filtering with a nove, more efficient version of dynamic sampling (Yu et al., 2025), see OLMoRL Infra for details.

- **Token-level loss**: We use a token-level loss to normalize the loss by the total number of tokens across the batch (Yu et al., 2025), rather than per-sample to avoid a length bias.

- **No KL Loss** We remove the KL loss as a common practice (GLM-4.5 Team et al., 2025; Yu et al., 2025; Liu et al., 2025b) as it allows less-restricted policy updates, and removing it does not lead to over-optimization or destabilized training.

- **Clip Higher**: We set the upper-bound clipping term in the loss to a slightly higher value than the lower bound to enable larger updates on tokens, as proposed by Yu et al. (2025).

- **Truncated Importance Sampling**: To adjust for differences between log probabilities from the inference and training engines, we multiply the loss by the truncated importance sampling ratio, following Yao et al. (2025).

- **No standard deviation normalization**: When calculating advantage, we do not normalize by the standard deviation of the group, following Liu et al. (2025b). This removes a difficulty bias, where questions with low standard deviation in their rewards (e.g. too hard or too easy) have their advantages significantly increased by the normalization term.

OlmoRL Formulation Our final objective function includes a token-level loss, truncated importance sampling, clip-higher, and no standard deviation in the advantage calculation:

$$\mathcal{J}(\theta) = \frac{1}{\sum_{i=1}^{G} |y_i|} \sum_{i=1}^{G} \sum_{t=1}^{|y_i|} \min\Big(\frac{\pi(y_{i,t} \mid x, y_{i,<t}; \theta_{\text{old}})}{\pi_{\text{vllm}}(y_{i,t} \mid x, y_{i,<t}; \theta_{\text{old}})}, \rho \Big) \min\Big(r_{i,t} A_{i,t}, \ \text{clip}(r_{i,t}, \ 1 - \varepsilon_{\text{low}}, \ 1 + \varepsilon_{\text{high}}) A_{i,t} \Big), \quad (1)$$

where $r_{i,t} = \frac{\pi(y_{i,t} \mid x, y_{i,<t}; \theta)}{\pi(y_{i,t} \mid x, y_{i,<t}; \theta_{\text{old}})}$, ε_{low} and $\varepsilon_{\text{high}}$ are the clipping hyperparameters. Here, $y_i \sim \pi_{\text{vllm}}(\cdot \mid x; \theta_{\text{old}})$ and $\pi_{\text{vllm}}(\cdot \mid x; \theta_{\text{old}})$ are the token probabilities returned from vLLM, ρ is the truncated importance sampling cap value (Yao et al., 2025), and the advantage $A_{i,t}$ for the t-th token t in the response y_i is calculated within the group G based on the relative reward of the outputs inside each group:

$$A_{i,t} = \Big(r(x, y_i) - \text{mean}\Big(\{ r(x, y_i) \}_{i=1}^{G} \Big) \Big). \quad (2)$$

$r(x, y_i)$ is the reward score returned by the corresponding verifier. Our hyperparameters for various runs are in Appendix Table 47.

Verifiers We extend verifiable rewards beyond math domains from OLMo 2 to include general domains. For each domain we use a different custom verifier (see Figure 16):

- **Math**: We use a rule-based verifier that performs basic normalization and compares with a reference answer with `SymPy` to determine answer correctness. The verifier returns 1 if the answer is determined the same as the reference answer and 0 otherwise.

- **Code**: We use a test-case based verifier that runs a set of test cases over the response. We experiment with (a) using the percentage of passed test cases as the reward and (b) returning 1 when the response passes all test cases and 0 otherwise.[25]

---

[24]We experimented with additional changes (e.g., overlong filtering), but did not find these gave consistent improvements.

[25]**Code Execution:** When performing RL on code environments, we need to actually execute the generated code against test cases to calculate our rewards. We use AWS Lambda to do so. Using a distributed cloud function approach ensures that verification does not block the trainer process, and allows us to scale seamlessly. Many test case suites, such as those present in SYNTHETIC-2 (PrimeIntellect, 2025), contain test cases designed to penalize programs with poor time complexity, and running these tests can exceed hundreds of MBs for a single program, exceeding the resources of our local machines.

Category	Prompt Dataset	# Prompts Used in Think RL	# Prompts Used in Instruct RL	Reference
Precise IF	IF-RLVR	30,186	38,000	Pyatkin et al. (2025)
Math	Open-Reasoner-Zero	3,000	14,000	Hu et al. (2025)
	DAPO-Math	2,584	7,000	Yu et al. (2025)
	AceReason-Math	6,602	–	Chen et al. (2025b)
	Polaris-Dataset	–	14,000	An et al. (2025)
	KlearReasoner-MathSub	3,000	9,000	Su et al. (2025c)
	OMEGA-train	15,000	20,000	Sun et al. (2025)
Coding	AceCoder	9,767	20,000	Zeng et al. (2025a)
	KlearReasoner-Code	8,040	–	Su et al. (2025c)
	Nemotron Post-training Code	2,303	–	NVIDIA AI (2025)
	SYNTHETIC-2	3,000	–	PrimeIntellect (2025)
General Chat	Tulu 3 SFT	7,129	18,955	Lambert et al. (2024)
	Wildchat-4.8M	7,129	18,761	-
	Multi-Subject RLVR	7,129	12,234	Su et al. (2025b)
Total	OLMO 3 Datasets	104,869	171,950	

Table 18 Breakdown of datasets in Dolci-Think-RL used for RL training. See §4.4.2 for further details on how each dataset was processed.

- **Instruction-following**: We pass the response through a set of functions that check adherence to a series of constraints from the prompt. A reward of 1 is assigned if all constraints are satisfied, and 0 otherwise.
- **Chat (reference)**: For tasks with a ground truth response, we pass the response to a LM judge to compare the model's response against a provided reference answer, and ask the judge to give a score in [0, 1] based on the quality of the response.
- **Chat (open-ended)**: We pass the response to a LM judge and ask the judge to give a score in [0, 1] based on the quality of the response without any reference answer.[26]

4.4.2 Dolci-Think-RL: Curating a State-of-the-art RLVR Dataset

We curate a large-scale and diverse dataset of roughly 100K samples across four domains: mathematics, coding, instruction following, and general chat to support robust RL on varied reasoning tasks while maintaining general utility. Each domain is associated with either a verifiable or non-verifiable reward signal (continuous or binary), ensuring that every instance can be automatically checked for correctness or general quality (see Figure 16). For all domains we take careful care of the provenance and licensing of sources. We provide the size of each dataset subsection after sourcing, filteringm and mixing in Table 18.

Step 1: Sourcing Prompts   In what follows, we will describe our data construction process.

- **Math** We combine community-curated math problems, including Open-Reasoner-Zero (Hu et al., 2025), DAPO-Math (Yu et al., 2025), AceReason-Math (Chen et al., 2025b), DeepScaler (Luo et al., 2025b), KlearReasoner-MathSub (Su et al., 2025c), and OMEGA (Sun et al., 2025) covering a wide range of mathematical domains including algebra, combinatorics, number theory, and geometry.

---

[26]Unless otherwise stated, for a LM-judge we host Qwen3-32B (Yang et al., 2025a) with thinking mode turned off using vLLM (Kwon et al., 2023), and allow a max input prompt of 32768 tokens while only allowing a response length of 2048 tokens. We provide the judge prompts in Figure 34 in the appendix. We additionally experimented with puzzle problem (checking if a puzzle solution is correct relative to a reference answer) and length-control (Aggarwal and Welleck, 2025) verifiers, but did not find it useful for downstream performance.

- **Coding** To construct reinforcement learning (RL) data for code, we required pairs of *(problem, test cases)*. We curate a diverse set of prompts for coding problems, including AceCoder (Zeng et al., 2025a), Klear-Reasoner Code (Su et al., 2025c), Nemotron Post-training Code (NVIDIA AI, 2025), SYNTHETIC-2 code (PrimeIntellect, 2025), and Open-Code Reasoner (Ahmad et al., 2025). We use the Klear-Reasoner and SYNTHETIC-2 test cases directly. For the other datasets, we run prompts through the following synthetic data pipeline: (1) *problem rewriting*, (2) *solution generation*, and (3) *test case generation*. After generating these triplets (problem, solution, test cases), we executed all model-generated or rewritten test cases against the corresponding solutions and kept examples with solutions that passed more than 80% of test cases while removing failed test cases. The resulting filtered dataset provided high-quality *(problem, test cases)* pairs suitable for training and experimentation with RL methods for code. We use the AceCoder prompts in function completion format, while all other datasets are in stdio format. Details of each step in code data synthesis pipeline can be found in Appendix A.5.3.

- **Instruction Following** We use the prompts from IF-RLVR (Pyatkin et al., 2025) with up to 5 constraints, which are sampled from IFEval (Zhou et al., 2023) and IFBench-Train (Pyatkin et al., 2025).

- **General Chat** We sample our general chat instances from three sources: (a) Tulu 3 SFT (Lambert et al., 2024); (b) the new Wildchat-4.8M data[27] containing a broad spectrum of user-chatbot interactions on ambiguous requests, code-switching, topic shifts, political debates, and more; and (c) Multi-subject-RLVR dataset (Su et al., 2025b) consisting of college-level English questions and objective answers written by domain experts for examination purposes. For Wildchat, we only sample from instances that are English, and do not require reasoning (such as math, and code). For Tulu 3, we first rewrote samples using GPT-4.1 for better clarity and to extract reference answers from the SFT set. We then generated 8 samples per prompt with a Qwen 2.5 7B model finetuned on OpenThoughts 2, and computed the F1 score between the reference answer and each response. We then remove all samples with average F1 score < 0.1 and > 0.8. This removes both noisy and difficult samples. Wildchat particularly has a high prevalence of role-playing and other character-based data. In order to balance the data, we filter any mention of a single character down to a maximum of 10 instances.[28] We then finally did some post-hoc manual filtering to remove code and math-centric prompts.

Step 2: Offline Difficulty Filtering As stated previously, to improve the sample efficiency of RL for our reasoner model, we generate eight rollouts for each prompt from the initial checkpoint of the model we train (e.g., if starting from the DPO-trained model, we generate from the DPO checkpoint). We then remove all samples that the model easily solves (that is, those with a pass rate greater than 62.5%). We sample with a temperature of 1.0 and top-p of 1.0, matching how we sample during RL training. We used offline filtering for the 7B OLMO 3 THINK, instead relying on active sampling and re-using the 7B DPO-filtered data for the 32B model due to compute and time constraints.

Step 3: Data Mixing When developing our data mixture and overall recipe, we found RL experiments were both long and compute-expensive, preventing us from ablating the full space of datasets and algorithmic choices. Instead, we established a pipeline in which: (a) we performed dataset-specific runs on an intermediate SFT checkpoint and observed downstream evaluation trends over the first 500-1000 RL steps; (b) focused on math domain training when testing new algorithmic changes; (c) periodically ran overall mixture experiments to ensure mixing was stable. When setting up our final run, we then took the most promising datasets, performed offline filtering, and carefully mixed them to ensure higher-quality datasets were upweighted, and roughly equal amounts of data were used for each domain (with slightly more focus on math and instruction following, as training on these domains seemed the most effective in per-dataset runs). Additionally, we downsample certain subtasks from OMEGA that the model especially struggled with based on offline filtering results.[29] We used this pipeline to development an RL mixture for the 7B model, and then simply used the same data mixture for the 32B due to compute and time constraints.

---

[27] https://huggingface.co/datasets/allenai/WildChat-4.8M

[28] In our intermediate general dataset of 57,819 samples, we found the top characters were 1. Natsuki: 1284 appearances, 2. Monika: 1243. 3. Sayori: 1076, 4. Yuri: 957, 5. Sakura: 453, and 6. MC: 424. All others were at 60 or lower before filtering.

[29] In particular, we downsample the following tasks by 50% *after* filtering: `trans_integrations`, `logic_gridworld_rookmove`, `logic_puzzles_grid_chip`, `comp_grid_chips`, `comp_n_gon`, `arithmetic_matrix_svd`, `comp_parametric_intersection`, `comp_-vertex_color`.

For our OLMO 3 THINK 7B training run, we used an initial version of our infrastructure without pipelineRL or truncation importance sampling, which took approximately 15 days. We later replicated the same run with our newer infrastructure, achieving similar performance in just 6 days of training.

4.4.3 OlmoRL Infrastructure in Open Instruct

We made substantial improvements on our reinforcement learning infrastructure to handle longer sequences and faster overall throughput. In RL, the key technical challenge for finetuning models that generate long sequences is managing inference – also called the rollouts. For our final models, we performed RL rollouts that were up to 32k tokens in length, and on average over 10k tokens (for the reasoner models). Inference dominated our costs, using 8 H100 nodes for training and 20 nodes for inference for the 32B OLMORL reasoner model. Given the cost of autoregressive inference, our learner spends 75% of the time waiting for data, so in terms of GPU utilization, we use approximately 5x as much for inference vs training. In fact, we use the minimal possible sharding configuration to fit the learner in memory, and don't prioritize speed at all, unlike in the supervised learning setting. For the 7B reasoner model, where we have less memory pressure on the learner, the situation was more dramatic, as we used 7 nodes for inference and only 2 for the learner. Given the similarly low utilization of the learner, we used approximately 14x as much compute for inference vs training. We suspect that we have a suboptimal sharding configuration for the 32B learner, and expect that we could do better in future work.

Fully Asynchronous Training Shown in Figure 17a, we employ an off-policy asynchronous RL setup (Noukhovitch et al., 2024) featuring a centralized learner distributed across multiple nodes via Deep-Speed (Rasley et al., 2020) and a large pool of actors, each running an independent vLLM (Kwon et al., 2023) instance. The learner produces prompts that are queued and dispatched to the actors, which execute the prompts, interact with the environment, and return results through a results queue that the learner uses to update the model parameters.[30] Due to the variance in completion length, a long time delta can emerge between completions in an individual batch of RLVR. The guiding principles to mitigate this issue is to make efficient use of resources (avoiding idling) and make processes asynchronous.[31]
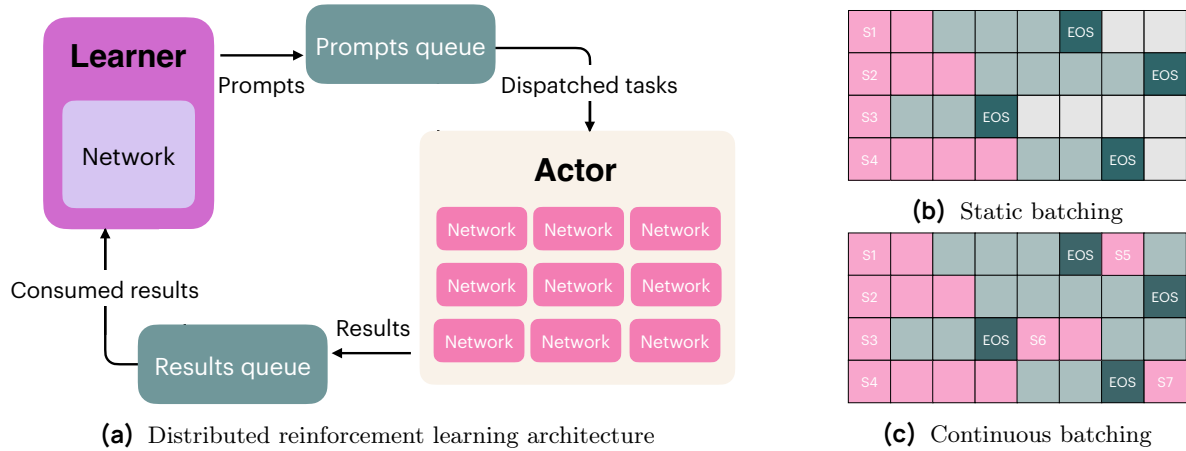
Continuous batching We employ continuous batching to constantly enqueue new generations as each one finishes to remove the compute waste for long generations (see Figure 17). This is in contrast to *static batching*, in which a batch of prompts are split over N actors, and each actor generates the entire batch[32], return the generated responses to the learner, and a new batch of data would be sent to each learner. Static batching is inefficient, as when one generation finishes that "slot" of the batch will remain empty until we get a new batch. The exact wasted compute can be calculated as the maximum sequence length minus the average sequence length divided by the maximum sequence length. With OLMo3, at a 32k generation length, we see a mean generation length of 14628 and a maximum of 32k, which means that up to 54% of our compute would have been wasted with static batching. See Figure 17 for an illustrated example.

Active Sampling To compensate for filtered instances, our fully asynchronous framework enables continuously pulling completions from the actor and resampling prompts into the queue. We actively sample and filter until we reach our desired batch size of non-zero gradient completions. Previously Yu et al. (2025) dynamic sampling would oversample and generate three times the number of prompts used in each training batch. This was to reasonably guarantee that the batch has enough completions with non-zero standard deviation. In contrast, our active sampling more efficiently uses the infrastructure. As demonstrated in section 6, we find this significantly stabilizes training and prevents batch size reducing over the course of training (a common issue with vanilla GRPO).

---

[30]For the 7B training runs, we use a single GPU for each actor and scale generation via data parallelism. The RL setup would be familiar to readers of Horgan et al. (2018) or Silver et al. (2017). For 32B, we use one node per actor and then similarly further scale via data parallelism.

[31]For one of our main RL runs, for instance, each training step averaged 1000 seconds, of which 125 seconds was spent running training. Each batched completion generation took 1000 seconds. As we overlap generation and training (Noukhovitch et al., 2024), the bottleneck is entirely generation. Consequently, significant engineering resources were spent improving the way generation is handled, where we could continue to use the training code used in OLMo 2, as we would need to speed up generation by > 8× for that to be a bottleneck.

[32]calling `llm.generate` in vLLM

(a) Distributed reinforcement learning architecture

(b) Static batching

(c) Continuous batching

Figure 17 **(a)** Distributed reinforcement learning architecture. **(b, c)** Static vs. continuous batching. Static batching wastes compute when generations have variable sequence lengths. Pink cells are prefilled tokens, green cells are decoded tokens, with dark green representing EOS. Grey indicates that sequence is not doing anything, so continuous batching backfills finished rows immediately (no wasted compute).

Inflight updates A common goal of RL training for LLMs is to minimize the degree of difference between the actor policy and the learner policy, i.e. minimize being off-policy (Van Hasselt et al., 2018). This can be achieved by synchronizing the weights after every training step as follows: each actor finishes all of their ongoing generations, dump the KV cache, and update their copy of the weights. However, this causes GPUs to be idle, and hurt training efficiency. Instead, we follow Piché et al. (2025) to immediately update the weights without pausing the engine, relying on the generation framework to be thread-safe, and continue generating, *without invalidating the KV cache*. This enables a significant increase in throughput: up to 4x faster with the same resources, without hurting accuracy.

Better Threading and Engineering Theses changes are primarily around handling the weight synchronization after each training step to make actors more efficient. Our new setup decouples the actors, allowing each one to start and stop by itself, without waiting for the rest of the actors to finish their syncs as well. Similarly, we make a large number of optimizations that were not machine learning specific, and were centered around efficiently using the *CPU*. For example, our initial implementation of continuous batching, for instance, was slower than static batching before adding a prefetch thread to our actors that constantly refilled the inference queue to see a throughput improvement.

Our final RL run ended up mixing carefully-filtered data from all domains roughly equally and running on top of the DPO checkpoint.

4.5 Key Findings

DPO yields gains where SFT on the same data cannot. Continued supervised fine-tuning directly on the chosen responses from DOLCI THINK DPO outright hurts the initial SFT model (Table 19), dropping all evaluation tasks. We conjecture that this is because the chosen responses (generated by Qwen3-32B Thinking) are weaker relative to data the model has already seen in DOLCI THINK SFT, and hence they are no longer useful targets for imitation. However, by pairing these chosen responses with rejected responses generated by an even weaker model, we construct a useful contrast, enabling preference tuning to drive strong gains beyond the initial SFT model (Table 19). Promisingly, these gains are not merely converting pass@k into pass@1 but rather expanding the reasoning frontier of the model (e.g., improved pass@k on AIME evaluations; Figure 20). These findings highlight contrastive learning with preference tuning as a useful stage for improving capabilities even when imitation is saturated.

	Subset of Olmo 3 Think Benchmarks									
Name	Avg.	MMLU	BBH	GPQA	Zebra	AGI	AIME25	AIME24	CHE	LCB	IFEval
Qwen3 32B (chosen)	83.2	88.8	90.6	64.7	78.2	90.2	71.0	80.3	90.9	89.6	87.4
Qwen3 0.6B (rejected)	35.1	55.8	41.5	27.22	29.8	59.2	15.2	11.2	14.8	34.4	62.3
Development 7B SFT	70.3	**76.1**	**83.9**	45.1	56.5	76.4	58.8	71.0	88.1	67.0	**79.7**
Cont. SFT on chosen	64.5	72.6	80.2	40.2	49.8	73.9	52.8	61.0	83.4	55.1	76.0
Delta learning	**72.9**	75.5	82.8	**48.4**	**60.9**	79.7	**66.3**	**75.7**	**91.5**	**72.6**	75.2
Development 32B SFT	77.7	**85.1**	89.2	**57.8**	71.8	85.5	66.7	73.4	89.5	75.3	**83.0**
Delta learning	**79.6**	**85.1**	**89.3**	57.1	**74.6**	**88.2**	71.7	75.3	**91.6**	**82.2**	81.3

Table 19 **The delta between chosen and rejected responses is critical**. Supervised finetuning directly on the chosen responses generated by Qwen3-32B Thinking hurts the Initial SFT model. In contrast, DPO tuning to prefer the 32B responses over weaker Qwen3-0.6B Thinking responses yields strong gains across math and code reasoning.

	Subset of Olmo 3 Think Benchmarks									
Name	Avg.	MMLU	BBH	GPQA	Zebra	AGI	AIME25	AIME24	CHE	LCB	IFEval
SFT	70.1	74.9	84.1	45.8	57.9	77.2	57.6	69.6	88.2	67.8	77.9
SFT + RLVR	71.8	77.1	82.5	44.6	61.6	78.4	60.3	71.0	87.9	69.4	**85.2**
SFT + DPO	72.7	74.8	83.7	**48.6**	60.6	79.1	62.7	**74.6**	91.4	75.1	75.9
SFT + DPO + RLVR	**73.8**	**78.0**	**85.3**	48.0	**62.4**	**79.7**	63.5	73.2	90.6	**75.2**	82.4

Table 20 **Delta learning provides a stronger initialization for subsequent RLVR than SFT alone.** We show the effect of conducting RLVR for 1000 steps after DPO and SFT on our 7B model on a subset of our evaluation suite. Note that here evaluations are from one run only. Preference tuning with delta learning first, followed by RLVR, yields the best overall performance. For RLVR, we use data offline-filtered by the corresponding starting point (SFT only or SFT + DPO).
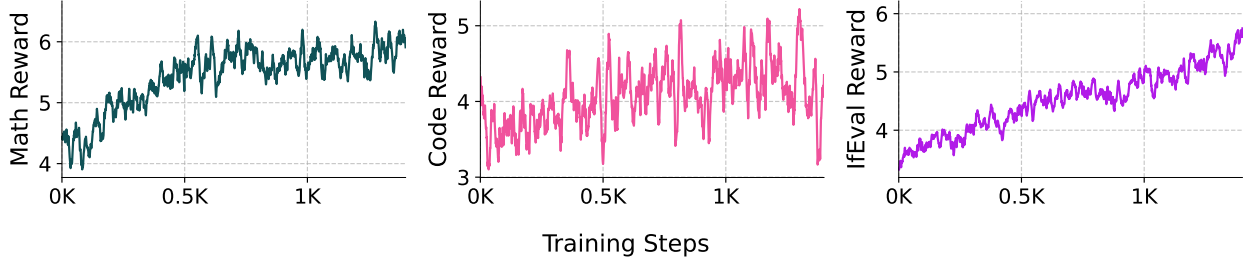
DPO and SFT benefit similarly from RL, but DPO remains a better starting point. Table 20 shows that running our final RL mix on the DPO model consistently yields better performance than running it on the SFT model. We find three primary differences, highlighted in Figure 19: for evaluations which RL does not improve, the DPO model often performs better and maintains its advantage during RL training (e.g., AlpacaEval). For evaluations explicitly targeted by RL (e.g., Omega), both the DPO and SFT models achieve similar end performance. For evaluations targeted by RL but hard to improve further from DPO (e.g. AIME 2025), the SFT model improves to get close to DPO performance. In no situation does the SFT model improve over the DPO model after RL, and as such we opt to focus on applying RL over our DPO model. Curiously, we find that the SFT model performs similarly when trained either with the data offline-filtered using the SFT or DPO model, suggesting that the additional samples filtered out (i.e., solved) by the DPO model do not provide additional signal for improving the SFT model. Further investigating this, we find that while the DPO model does display lower entropy, it in fact has higher Pass@K performance on AIME evaluations, as shown in Figure 20. This suggests that the DPO model remains a strong starting point for RL relative to the SFT model, since RL tends to convert Pass@K performance into Pass@1 performance (Yue et al., 2025).

Rewards steadily increase across all domains during RL. Figure 18 plots per-verifier reward curves along with average output length. We find that reward steadily increases across all domains , albeit at differing rates (with instruction-following data increasing most steadily, and code reward increasing most slowly). We plot more RL curves in the appendix (Figure 35). Interestingly, we find that sequence lengths first slightly dip, and then slowly increase over time. This is likely due to the reasoning SFT and DPO already training the model to produce length reasoning traces of up to the maximum response length of 32k tokens.

Mixing RL data from varied domains can prevent over-optimization. Figure 20 (left) demonstrates that training on specific domains can lead to over-optimization, in which its performance on evaluations outside that domain drops, while training on a mix keeps steady improvements across different domains. For example, we observe a trade-off when performing OLMORL on IFEval alone, wherein higher IFEval scores correlate

	Total tokens (Mtok)	Tokens/second	MFU (%)	MBU (%)
OLMo 2	6.34	881	0.30	12.90
+ continuous batching	7.02	975	0.33	14.29
+ better threading	9.77	1358	0.46	19.89
+ inflight updates (OLMo 3)	21.23	2949	1.01	43.21

Table 21 Effect of core infrastructure improvements to OlmoRL. We ablate the effect of each component by measuring the training speed (tokens/second) and utilization metrics as we add each component in turn from the original OLMo 2 RL infrastructure. Inflight updates has the most drastic improvement.

Figure 18 Reward curves during training of Olmo 3 Think 7B. Average, math, code, and IF reward over RL training for the final RLVR training run of OLMo 3 THINK. Reward steadily grows across domains, suggesting smooth training. See Figure 35 for further RL curves.
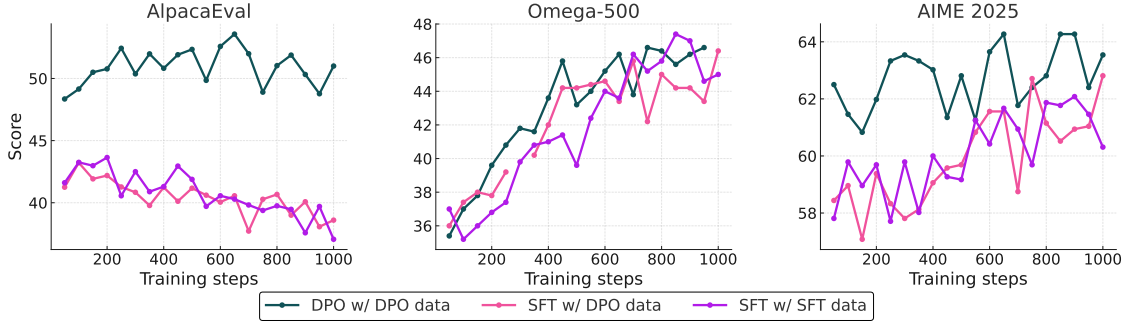
with lower lower AlpacaEval scores. However, when we perform our final mixed training, we are able to maintain high AlpacaEval scores without compromising IFEval performance, as the LM judge reward ensures that the model continues to produce well-formed chat responses.

Mixing data yields lower train reward, but not lower downstream performance. While Figure 20 demonstrates that our final mixture run achieves downstream performance similar to or greater than RL training runs on single domains, we find that we observe *lower* train reward across each domain when training on mixed data as opposed to single-domain data, as seen in Figure 21. This suggests that mixing data may in fact reduce the model **over-optimizing during training**, preventing some degree of reward-hacking, and thus generalizing better to downstream evaluations. This may explain why RL training on broader data mixtures can outperform domain-specific mixtures (Cheng et al., 2025).
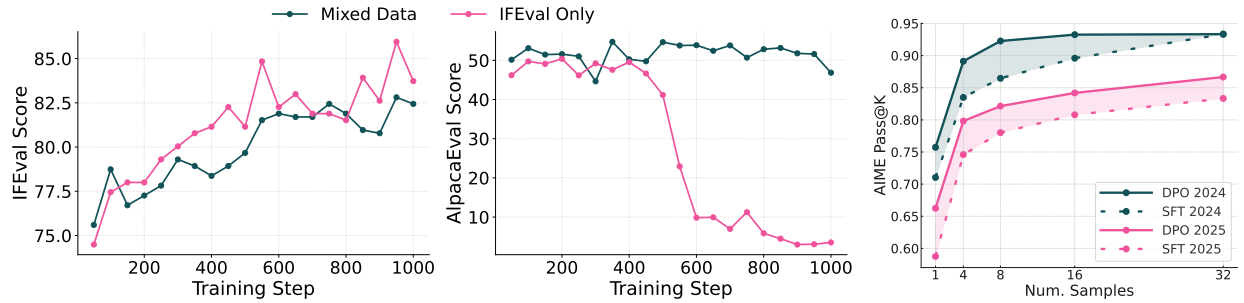
Continuous batching and inflight updates are crucial to training speed. Using a reasoner SFT or DPO as a starting point stresses RL training to its limits, as the model starts with extremely long average generation lengths. Table 21 demonstrates how using continuous batching and inflight updates is crucial to training speed, allowing us to achieve two times faster training on half as many GPUs, making experimentation and long RL runs more tractable.[33] To carefully benchmark this, we ablate the changes to our RL infrastructure between OLMo 2 and OLMo 3. See Table 21. For each ablation, we ran a benchmark experiment for 2 hours using 2 8x A100 nodes. One node was used for training, and one for inference. Since inference is our bottleneck, we report Model FLOPs Utilization (MFU) and Model Bandwidth Utilization (MBU) based solely on the single node used for inference. A typical full-scale experiment would use many more nodes for inference, typically with a 8:1 ratio (or more) of inference nodes to training nodes. The benchmark experiment generates a batch of 128 completions for each training step, using 64 prompts, each sampled twice, with a maximum output length of 32000, and a maximum input length of 2048, leading to a context length of 2048.[34]

---

[33]While an initial checkpoint took 14 straight days of training across 9 nodes to achieve 1 epoch, with continuous batching and inflight updates, we could achieve 1 epoch on 5 nodes in 7 days.

[34]Script can be found in the https://github.com/allenai/open-instruct, at `scripts/benchmarking/olmo3_infra.sh`.

Figure 19 Using DPO as a starting point for RLVR works best. AlpacaEval, Omega-500, and AIME 2025 performance over the course of RLVR training when starting from OLMO 3 7B SFT or DPO, training using either data filtered via the DPO model (w/ DPO data) or SFT model (w/ SFT data). The importance of starting from DPO or SFT depends on the evaluation, but starting from DPO is overall preferable.
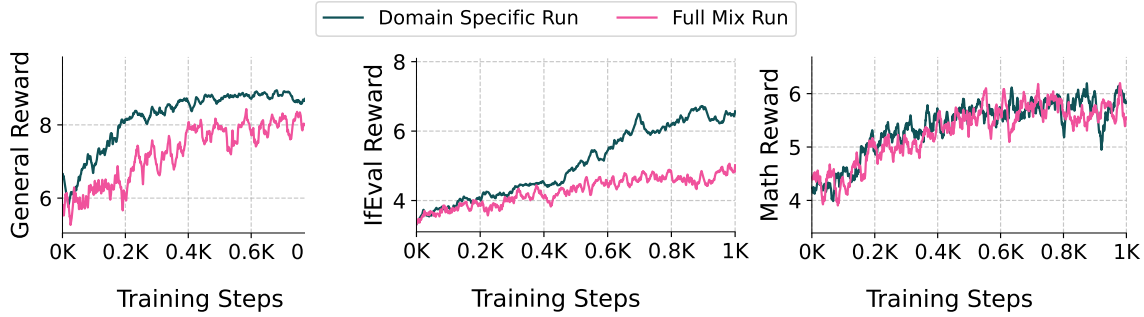

Figure 20 Training on mixed data prevents overfitting (left). We plot IFEval and AlpacaEval performance over RL training on OLMO 3 THINK SFT 7B when training on IFEval data only or on mixed data. Training on mixed data achieves similar IFEval performance whilst maintaining high AlpacaEval performance. **DPO displays higher pass@K performance than SFT (right).** We plot pass@K for AIME 2024 and 2025 for SFT and DPO thinking models for up to K=32. DPO consistently improves performance, even at higher K.

Olmo 3-RL shows significant improvement in precise instructino following. Instruction following performance The precise instruction following performance increases across post-training stages, with the final RL training stage leading to the biggest improvements in Olmo3's precise instruction following abilities, as shown in Table 22, for both the development (IFEval) and the unseen (IFBench) evaluations.

		Think-SFT	Think-DPO	Think-RL	Instruct-SFT	Instruct-DPO	Instruct-RL
7B	**IFEval**	83.9	80.6	**86.0**	81.7	82.0	**85.77**
	IFBench	30.0	28.3	**41.6**	27.4	29.3	**32.3**
32B	**IFEval**	83.7	82.3	**88.0**	-	-	-
	IFBench	37	34.4	**47.6**	-	-	-

Table 22 Summary of precise instruction following results on IFEval and IFBench, for both the Olmo3 Think and Olmo3 Instruct models (at 7B and 32B sizes), across various stages of the post-training pipeline.

Figure 21 Per-domain training yields higher train rewards. We plot the train reward over RL training for per-domain and overall mix (i.e., final) training runs. In each plot, we train an intermediate SFT model using RLVR with data only from general, IF, and math subsets, and compare to training on our overall mix. Whilst the domain-specific runs achieve higher train reward, Figure 20 shows this does not necessarily yield improved downstream performance.

Skill	Benchmark	Olmo 3 Instruct 7B-SFT	Olmo 3 Instruct 7B-DPO	Olmo 3 Instruct 7B	Qwen 3 8B (no reasoning)	Qwen 3 VL 8B Instruct	Qwen 2.5 7B	Olmo 2 7B Instruct	Apertus 8B Instruct	Granite 3.3 8B Instruct
Math	MATH	65.1	79.6	87.3	82.3	91.6	71.0	30.1	21.9	67.3
	AIME 2024	6.7	23.5	44.3	26.2	55.1	11.3	1.3	0.5	7.3
	AIME 2025	7.2	20.4	32.5	21.7	43.3	6.3	0.4	0.2	6.3
	OMEGA	14.4	22.8	28.9	20.5	32.3	13.7	5.2	5.0	10.7
Reasoning	BigBenchHard	51.0	69.3	71.2	73.7	85.6	68.8	43.8	42.2	61.2
	ZebraLogic	18.0	28.4	32.9	25.4	64.3	10.7	5.3	5.3	17.6
	AGI Eval English	59.2	64.0	64.4	76.0	84.5	69.8	56.1	50.8	64.0
Coding	HumanEvalPlus	69.8	72.9	77.2	79.8	82.9	74.9	25.8	34.4	64.0
	MBPP+	56.5	55.9	60.2	64.4	66.3	62.6	40.7	42.1	54.0
	LiveCodeBench v3	20.0	18.8	29.5	53.2	55.9	34.5	7.2	7.8	11.5
IF	IFEval	81.7	82.0	85.6	86.3	87.8	73.4	72.2	71.4	77.5
	IFBench	27.4	29.3	32.3	29.3	34.0	28.4	26.7	22.1	22.3
Knowledge	MMLU	67.1	69.1	69.1	80.4	83.6	77.2	61.6	62.7	63.5
QA	PopQA	16.5	20.7	14.1	20.4	26.5	21.5	25.5	25.5	28.9
	GPQA	30.0	37.9	40.4	44.6	51.1	35.6	31.3	28.8	33.0
Chat	AlpacaEval 2 LC	21.8	43.3	40.9	49.8	73.5	23.0	18.3	8.1	28.6
Tool Use	SimpleQA	74.2	79.8	79.3	79.0	90.3	78.0	-	-	-
	LitQA2	38.0	43.3	38.2	39.6	30.7	29.8	-	-	-
	BFCL	48.9	49.6	49.8	60.2	66.2	55.8	-	-	-
Safety	Safety	89.2	90.2	87.3	78.0	80.2	73.4	93.1	72.2	73.7

Table 23 Overview of Olmo 3 Instruct-7B results on Olmo 3 Eval. suite. All numbers are the mean of three runs.

5 Olmo 3 Instruct

Recent studies suggest that real-world language model use predominantly centers around general tasks such as advice seeking and information recall (Chatterji et al., 2025) that may not require extensive reasoning. Hence, OLMO 3 INSTRUCT was constructed with these real use-cases in mind, as users expect chat models to quickly and helpfully respond to common queries. Everyday chat settings often do not require the inference-time scaling of OLMO 3 THINK, allowing us to be more efficient at inference time on common tasks by not generating extended internal thoughts.

These different model types demand different data to support them. We focus on improving the interactability of the models by introducing multi-turn DPO data and promoting concise responses in our delta-learning preference-tuning pipeline. Additionally, OLMO 3 INSTRUCT is trained for function-calling, for which we release new SFT datasets. Together, our recipe yields OLMO 3 INSTRUCT models that effectively leverage tools and efficiently responds to user queries. See Table 23 for the full set of our evaluations and comparisons with similar models. The benchmarks used for all the evaluations except function-calling are the same as those used for OLMO 3 THINK (Section 4.1).

Dataset	Env. interactions	# Trajectories	# Unique functions	% Multi-turn	% Multi-step
Science QA	Real (MCP)	22.6K	8	-	42.3%
Web Search QA	Real (MCP)	6.6K	3	-	76.1%
SimFC	Simulated	200K	42.6K	42.3%	23.8%

Table 24 Details of function calling datasets. Multi-turn refers to multiple user turns per trajectory and multi-step refers to multiple environment interactions per user request.

5.1 Main Results for Olmo 3 Instruct

Table 23 demonstrates the results of OLMO 3 INSTRUCT on our evaluation suite. OLMO 3 INSTRUCT significantly outperforms Qwen 2.5-7B Instruct, Olmo 2 7B Instruct, and Apertus 8B Instruct.

5.2 Supervised Finetuning with Dolci Instruct SFT

We construct DOLCI INSTRUCT SFT by building upon OLMO 2 INSTRUCT, making significant improvements to advance general chat, reasoning, and function calling capabilities.

5.2.1 New function-calling training data

Our goals for curating tool use training data for OLMO 3 INSTRUCT are to provide the model a strong foundation in basic function-calling and to expose the model to trajectories demonstrating the effective use of real environments (i.e., MCP servers) to perform tasks. Accordingly, we collect two kinds of trajectories synthesized using LLMs, described below.

Trajectories with real interactions We collect trajectories demonstrating agents' use MCP servers to answer queries in the following two domains. All these trajectories have single user turns and multiple agent-environment interactions.

- **Science QA Dataset** contains two broad classes of queries requiring retrieval and reasoning over scholarly content: (i) paper content-based queries, which focus on information present in the abstract or full text of papers and (ii) citation graph-based queries, which are about metadata such as authors, venues, and citations. Trajectories associated with the queries were obtained using an agent based on GPT-4.1-mini equipped with the ASTA Scientific Corpus (ASC) MCP server[35], which provides structured access to metadata and paper content on Semantic Scholar[36]. Additional details about these datasets are provided in Appendix A.5.2.

- **Web Search QA Dataset** is generated from a multi-stage process that combines benchmark-derived and real-world queries. Queries are drawn from open-access benchmarks: HotpotQA (Yang et al., 2018), TaskCraft (Shi et al., 2025), and WebWalkerQA (silver) (Wu et al., 2025a), as well as from consented, publicly released user prompts from SearchArena (Miroyan et al., 2025) and OpenScholar (Asai et al., 2024). We filtered the set of queries using GPT-5 to keep only those that require search, and whose responses need to be long-form and are verifiable. The trajectories for these queries were obtained from a GPT-5 agent equipped with the Serper API[37] which provides access to a Google search tool and a tool for fetching webpages given their URLs. Additional details about query filtering and trajectory generation can be found in Appendix A.5.2.

Trajectories with simulated interactions While training on trajectories with executable environments is expected to teach the model to effectively deal with real environment outputs and handle unexpected errors, it is difficult to curate such trajectories at scale, thus potentially limiting the model's generalization to unseen

---
[35] https://allenai.org/asta/resources/mcp
[36] https://www.semanticscholar.org/
[37] https://serper.dev/

tools at inference-time. To fill this gap, we also create a dataset of synthetic trajectories with LLM-simulated environments which are much easier to scale. We call this dataset **SimFC**. We start with a large pool of tool sets or APIs from existing datasets (e.g. xLAM (Liu et al., 2024c), ToolACE (Liu et al., 2024b)), and from publicly available MCP servers, and prompted LLMs (GPT-4o, GPT-4.1, and GPT-5) to generate entire trajectories including simulated user queries, environment responses, and assistant messages. We design prompts to ensure the dataset contained a variety of interaction patterns including multi-turn, multi-step, and refusals due to inadequate information or tools. Additional details about this dataset and illustrative prompts used for generation can be found in Appendix A.5.2 and Figures 36, 37.

Balancing function diversity with interaction complexity   As illustrated by the statistics in Table 24, the two types of trajectories have key differences. SimFC has a large number of trajectories with diverse sets of functions. We found that synthesizing trajectories with multiple user turns (multi-turn trajectories) was relatively easier than those with multiple assistant-environment interactions per user request (multi-step trajectories). However, the latter class usually corresponds to more complex tasks. On the other hand, the MCP-based datasets, while smaller in size, are naturally more complex in terms of multi-step interactions.

Unified data format   Across all tool-use data, we adopt consistent tool definition and tool calling formats which we found to be crucial for stable and high-quality tool-use behavior. Particularly, we use the OpenAPI specification[38] for all tool definitions and represented all function calls as pythonic code blocks. We provide tool specifications in the system prompt, encapsulated tool calls with XML tags within the assistant role, and presented environment outputs to the model within a special environment role. We also extend the tokenizer's vocabulary with dedicated special tokens corresponding to these tags, which we found was essential for tool use training.

Evaluating function calling   We evaluated the function calling capabilities of OLMO 3 INSTRUCT in terms of *intrinsic function calling* and *extrinsic task completion* accuracies using different benchmarks. We used the Berkeley Function Calling Leaderboard (BFCLv3) (Patil et al., 2025) to evaluate intrinsic function calling accuracy. This benchmark focuses on models' ability to choose the relevant functions and the right values for their arguments to accomplish a given task in settings that require one or more interactions with simulated users and environments. We evaluated the task completion accuracy of OLMO 3 INSTRUCT in comparison with similar models when they are deployed as agents with access to tools served via Model Context Protocol (MCP) servers. Particularly, we used the Asta Scientific Corpus (ASC) tool (Bragg et al., 2025) that serves eight functions for accessing scientific literature, and the Serper API which provides Google search tool and web browsing functionalities. To evaluate models' usage of the ASC tools, following Bragg et al. (2025), we use a subset of 75 questions from LitQA2 (Skarlinski et al., 2024) for which the associated papers can be found in ASC's index. We evaluate the models' usage of search and browsing tools using a subset of SimpleQA[39] (Wei et al., 2024).

We use the official Gorilla repository[40] for BFCLv3 evaluations. For LitQA2 and SimpleQA, we implement a basic function calling agent using OpenAI's Agent SDK. This agent uses the tools provided by the relevant MCP server, and interacts with the environment by iteratively making function calls and processing the outputs of executing them to solve the given tasks. For LitQA2 and SimpleQA, We also measure model performance when deployed in a *No-Tools* setting, in which we provide no tools to the agents and they are expected to solve the tasks entirely from the models' parametric knowledge. We use a zero-shot evaluation for all these benchmarks. We sample from models at temperature 0 and for LitQA2 and SimpleQA, allow the agents at most 10 turns to finish each task. We run each evaluation three times and report the average accuracy. We release our code[41] for running our MCP-based tool use evaluations.

---

[38] https://swagger.io/specification/
[39] https://huggingface.co/datasets/akariasai/sampled_simpleqa
[40] https://github.com/ShishirPatil/gorilla
[41] https://github.com/allenai/mcp-tool-eval

Name	Subset of Olmo 3 Instruct Benchmarks								
	Avg.	MMLU	BBH	GPQA	MATH	GSM8K	CHE	AE	IFEval
Base mix	29.0	50.0	29.5	25.2	6.6	30.1	23.2	5.8	61.7
Base mix + Aya	29.1	51.9	28.2	28.1	6.9	31.4	21.3	4.9	60.3
Base mix + Code	28.7	51.1	28.8	25.0	6.9	28.2	26.8	5.8	57.3
Base mix + Flan	30.3	51.9	35.0	26.8	6.6	34.7	21.3	5.8	60.3
Base mix + IF	30.7	51.4	24.7	25.5	7.9	42.2	14.6	5.5	74.1
Base mix + Math	29.3	49.9	23.9	29.2	14.2	39.7	18.3	5.4	54.0
Base mix + Safety	27.0	51.7	28.3	24.8	6.5	28.2	14.0	6.8	56.0
Base mix + Science	29.4	53.4	25.3	28.1	8.3	34.9	20.7	6.8	57.3
Base mix + Wildchat	30.9	51.9	30.7	23.7	6.9	32.2	23.2	19.2	59.7

Table 25 Results of our instruct SFT mixing ablations on top of OLMo 2.

Name	Subset of Olmo 3 Instruct Benchmarks									
	Avg.	BBH	GPQA	MATH	GSM8K	OMEGA	CHE	MBPP	LCB	AE	IFEval
No thinking SFT	44.5	46.5	29.7	60.3	87.6	8.6	63.8	54.1	13.0	27.0	81.0
Thinking SFT	47.8	46.6	34.4	65.9	91.1	12.2	68.7	57.1	17.1	27.1	84.7
Gain from thinking SFT first	3.3	0.1	4.7	5.6	3.5	3.6	4.9	3.0	4.0	0.1	3.7

Table 26 Results of training an intermediate OLMo 3 INSTRUCT checkpoint with and without thinking SFT first.

5.2.2 Curating Dolci Instruct SFT

Step 1. Sourcing Prompts and Completions   Our prompt collection includes all our new function calling data (Section 5.2.1), new prompts for instruction following §4.2.1 and science, and more chat prompts from WildChat (Zhao et al., 2024a). For examples that we include that originally contained reasoning traces (such as the OpenThoughts3 science subset described in § 4.2.1), we remove the reasoning traces and special tokens. We also update completions from older models such as GPT-3.5 and GPT4 with completions from GPT-4.1. We show a summary of our instruct SFT mix in Table 27.

Step 2: Filtering & Mixing   We follow the same filtering and mixing procedure detailed in Section 4.2.1. For OLMo 3 INSTRUCT, our base mix is 100k examples from an updated intermediate mix based on the OLMo 2 SFT mix. We show results of our data mixing experiments on OLMo 2 in Table 25.

Starting from Olmo 3 Think SFT   We train the SFT stage of OLMo 3 INSTRUCT starting from the OLMo 3 THINK SFT model as shown in Figure 2. We found that this significantly improves the performance of the Instruct model, as shown by the results in Table 26.

5.3 Preference Tuning with Dolci Instruct-DPO

We create DOLCI INSTRUCT-DPO by extending the strong base of our delta-learning heuristic preference pipeline (Section §4.3) with further preference signals to enhance the our model's behavior in general use settings. For example, we enrich our data with contrastive pairs from an improved GPT-judge pipeline for general alignment. Additionally, user interaction with LMs commonly requires multi-turn capabilities, so we introduce synthetic multi-turn conversations to our preference data. We also observe that preference-data pipelines often promote overly verbose responses; we introduce counteracting interventions to promote brevity in model responses by mitigating length bias in the preference data.
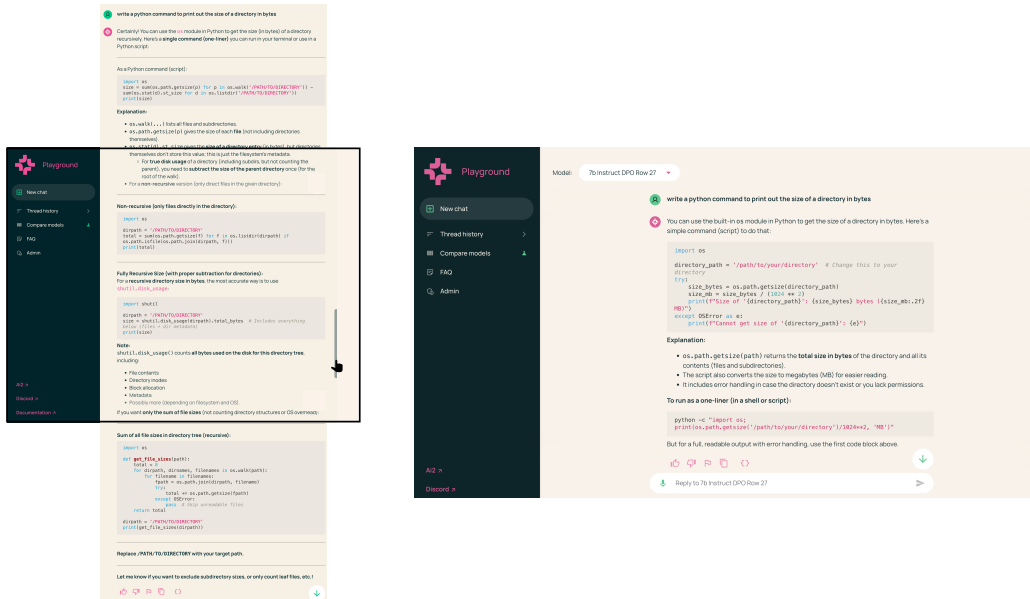
5.3.1 Preference Signals

DOLCI INSTRUCT-DPO is constructed from a composite of several preference signals to promote model capabilities and general usability:

Category	Prompt Dataset	# Prompts used in SFT	# Prompts used in DPO	Reference
Chat & Precise IF	WildChat	302,406	30,248	Zhao et al. (2024a)
	DOLCI INSTRUCT Precise IF	136,833	35,057	–
	DOLCI INSTRUCT Persona Precise IF	–	6667	Lambert et al. (2024)
	OpenAssistant	7,132	493	Köpf et al. (2024)
Math	Tülu 3 Persona MATH	149,958	14,728	Lambert et al. (2024)
	Tülu 3 Persona Algebra	19,999	2,025	Lambert et al. (2024)
	Tülu 3 Persona GSM	49,980	5,011	Lambert et al. (2024)
	OpenMathInstruct 2	50,000	5,325	Toshniwal et al. (2024)
Coding	DOLCI INSTRUCT Python Algorithms	186,345	24,096	–
	Tülu 3 Persona Python	34,999	4,598	Lambert et al. (2024)
	Evol CodeAlpaca	107,270	12,953	Luo et al. (2023)
Safety	CoCoNot	10,957	2,203	Brahman et al. (2024)
	WildGuardMix	49,373	12,037	Han et al. (2024)
	WildJailbreak	49,965	12,431	Jiang et al. (2024)
Science	SciRiff	4,557	8,874	Wadden et al. (2024)
	DOLCI INSTRUCT OpenThought3+ Science	99,268	26,134	Guha et al. (2025a)
Multilingual	Aya	99,987	6,523	Singh et al. (2024)
Other	TableGPT	5,000	1,218	Zha et al. (2023)
	FLAN	89,981	16,120	Wei et al. (2021)
	Logic Puzzles	159,882	–	
	Verifiable Reasoning	310,572	–	
	DOLCI INSTRUCT Hardcoded	69	–	–
	DOLCI INSTRUCT Tool Use	227,579	–	
Multiturn	DOLCI INSTRUCT Self-Talk	–	5,000	–
	DOLCI INSTRUCT Synthetic Context	–	5,000	–
Not used in SFT	DaringAnteater	–	878	Wang et al. (2024b)
	Ultrafeedback	–	22,303	Cui et al.
Total	OLMO 3 Datasets	2,152,112	259,922	

Table 27 OLMO 3 INSTRUCT prompt sources for both SFT and DPO.

Delta-learning heuristic pairs. Similar to DOLCI THINK-DPO, we construct heuristic contrastive pairs by generating chosen responses with a large model (Qwen3-32B) and rejected responses with a small model (Qwen3-0.6B) following Geng et al. (2025). Note that we turn off thinking mode, as we do not need internal thinking traces.

Delta-aware GPT-judged pairs. We additionally generate GPT-judged preference pairs to add a further source of preference signal. Our initial attempts to modernize the Ultrafeedback pipeline from OLMO 2 and Tülu 3 3 by improving the quality of the LLM judge (GPT-4o → GPT-4.1) and updating our data-generator model pool failed to yield gains and even hurt relative to the OLMO 2 preference dataset baseline. We conjectured that this is because the majority of our data generators were too high-quality, and hence on average there was minimal meaningful contrast between the resulting chosen and rejected pairs. To mitigate this, we explicitly introduce interventions designed to *lower* the quality of the rejected response. We (1) ensure that responses

Figure 22 Length control promotes concise, usable responses. On the left is a response from a model trained without length control; on the right, with length control. Promoting brevity in model responses makes the response easier to use and understand.

from weaker models are always present in the response set we judge for each prompt, and (2) select the *worst* response as the rejected completion to maximize the resulting delta. We find these "delta-maximizing" interventions to be critical for the quality of preference pair data; see our findings in Section 5.5 for details.

Multi-turn preferences. To ensure OLMO 3's usability in realistic multi-turn conversations, we further added a multi-turn preference dataset with prompts synthetically extended from the Tülu 3-DPO dataset. Preference pairs differed in only the last turn of the conversation to avoid ambiguity in quality ranking between turns of the same conversation. Synthetic conversations were generated with two methods: (1) **self-talk** extending the original prompt into a multi-turn conversation with LLM-generated follow-up requests and (2) **synthetic-context** created by generating related, independent questions or paraphrases of the initial prompt to use as previous user turns with associated completions. The combination of these generation methods ensures diversity in generated conversations. Final turns were generated with the delta-learning heuristic (Geng et al., 2025) where chosen/rejected completion pairs were generated by either GPT-4o and GPT-3.5 or Qwen3-32B and Qwen3-0.6B (no-thinking) respectively.

Controlling length bias Preference data often has a length bias: the chosen responses are significantly longer than the rejected responses. This comes from sourcing synthetic response pairs where more information is treated more helpful by both LLM judges and heuristic pairs. Namely, LLM judges such as the GPT judge in our pipeline tend to prefer longer responses. Similarly, we empirically observe that preference pairs made with the delta-learning heuristic also exhibit length bias; larger models generate longer responses (Figure 23). Thus, in addition to useful quality signal, this length bias in the preference data is often learned by the model during preference tuning, after which its generation length per prompt increases significantly. Since excessive verbosity can be undesirable for common real-use settings (see an example in Figure 22), we filter the chat and multi-turn subsets of our preference data to limit the length difference between the chosen and rejected responses to 100 tokens.

5.3.2 Prompt Mixing

Our prompt pool for GPT-judged and delta-learning heuristic pairs (see Table 27) is derived from the DOLCI INSTRUCT SFT dataset supplemented with the DaringAnteater and Ultrafeedback subsets from the OLMO

2-7b preference dataset. Because DPO performance is not monotonic with more data (see Figure 23), we optimize the prompt distribution as a ratio within a set data budget, and treat dataset size as a hyperparameter when training.

To determine our final preference-tuning prompt distribution, we begin with near-uniform random sampling[42] of 100k examples as an empirically strong baseline prompt mix. We then perform ablations of prompt-domain subsets to determine the impact of prompts of each domain. Additionally, we perform experiments that paired 50k samples of our base mix with 50k samples from a given domain, allowing us to understand the effects of upsampling each prompt domain.

Notably, prompt-domain distributions do not consistently align with the *contrast* exhibited in the pair and thus improvements in the corresponding downstream evaluation domains. For example, upsampling code prompts led to the unintuitive effect of decreasing code benchmark performance (see Table 46 in the Appendix). For our determining our final mix, we created 9 candidate mixes based on expert intuition gained from our ablations, comparing these hand-crafted mixes against the uniform sampling baseline. Our final mix was determined empirically; we find that hand-crafted mixes generally slightly outperformed random sampling.

5.3.3 Training

We follow the same setup of OLMO 3 THINK and sweep the same hyperparameters, namely learning rate and dataset size. We further sweep different length control interventions by creating datasets with differing token cutoffs for length filtering. We select the best performing checkpoint of each length budget, and then select the final OLMO 3 INSTRUCT-DPO checkpoint based on qualitative vibe tests and performance-vs-length analysis.

5.4 Reinforcement Learning with Dolci Instruct-RL

We use the same pool of prompts as DOLCI THINK-RL (§4.4.2) with the exception of (i) utilizing less challenging datasets in math and code domains, (ii) and skipping the offline difficulty filtering as our instruct model focuses more on general instruction following rather than complex reasoning.

5.4.1 Training

Following our OLMO 3 THINK recipe, we train OLMO 3 INSTRUCT on a mixture of general chat, math, and code data.[43] We likewise employ OLMORL for training, with a maximum response length of 8K tokens. Since our goal for OLMO 3 INSTRUCT is to avoid generating excessively long outputs and preserve general usability, we apply RL on top of two DPO candidates: one that achieved the best average performance, and another with slightly lower performance but better qualitative "vibe-test". We then choose the final RL checkpoint based on final average performance, length analysis, and "vibe-test". Concretely, we begin by ranking checkpoints by average score; in the case of ties, we place more emphasis on datasets that do not scale with test-time compute (*e.g.,* MATH and AIME) to avoid biasing our selection towards models with overly long responses. Finally, we apply the "vibe-test" to identify regressions or undesirable behaviors that may fall outside the scope of our evaluation suite.

5.5 Key Findings

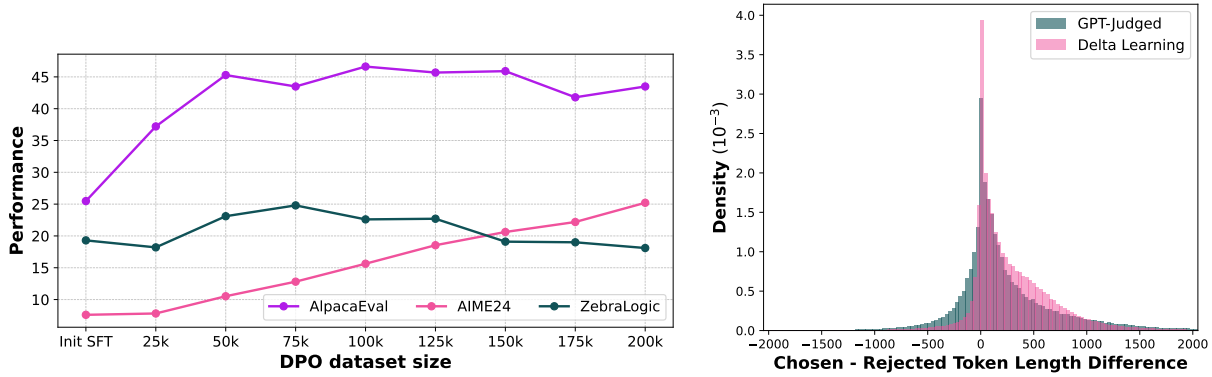Below, we summarize our key findings across all 3 stages of OLMO 3 INSTRUCT training:

Starting from the Olmo 3 Think SFT is helpful We find that training our instruct model on top of the thinking SFT model both increases model performance on benchmarks, as shown in Table 26, and also does not increase average model response length.

---

[42]We decided early to truncate the number of Wildchat prompts to be at most 35% of the prompt mix. If you read Wildchat prompts for a month, you would too.

[43]Preliminary experiments indicated that alternative RL setups—for example, first warming up on math-only data and then switching to a mixed dataset without math—resulted in suboptimal performance.

Figure 23 Ideal preference dataset size depends on the downstream task (left). Both AlpacaEval and ZebraLogic performance increases up to around 75-100K samples, beyond which further data scaling hurts or does not help. In contrast, AIME2024 does not saturate before the point at which AlpacaEval and ZebraLogic begin to see drops in performance. Hence, to strike an ideal balance between all downstream tasks, we sweep dataset size as a hyperparmeter during training. **Unfiltered preference data exhibits a length bias (right).** A significant portion of the distribution has longer chosen than rejected completions. For example, the 80th percentile of token difference for the GPT-judged data is 538 tokens and for the delta-learning heuristic pairs is 564 tokens.

High contrast in preference pairs drives DPO improvements. We observe that a high contrast between completions is critical for achieving improvements during DPO training (Table 29). Using LLM-judge pipelines requires carefully thinking about maximizing the delta between chosen and rejected responses. Our initial attempts to modernize the OLMo 2 preference data pipeline by improving the models used to generate responses failed to yield any improvements beyond the OLMo 2 data baseline (Table 29). This is because the models used for synthetic completions were too good: the chosen and rejected responses no longer had meaningful contrast because they were both from good models. Extending the finding from delta learning that high contrast pairs are critical for performance, we introduced interventions to explicitly lower the quality of the rejected response and therefore increase the magnitude of the quality delta between chosen and rejected response pairs. These resulting **delta-aware** GPT pairs significantly outperform the OLMo 2 preference data.

Combining different preference signals improves overall performance. We combine delta learning with GPT-judged preference data to get the "best of both worlds." Empirically, tuning with delta learning or GPT-judged pairs yield a different spread of gains; we find that these gains are complementary. Combining both sources of preference signal outperforms using either alone (Table 29).

The ideal amount of preference data depends on the downstream task. DPO performance peaks after different amounts of training for different downstream task domains. We plot preference tuning performance for example tasks across varying amounts of delta-learning heuristic pairs[44] in Figure 23. Further optimization beyond these optimal points hurts downstream performance, consistent with theoretical results showing that early stopping is important for DPO (Geng et al., 2025; Azar et al., 2023). Practically, this informs our training approach: we sweep LR and dataset set size to control the amount of total optimization, and pick the best performing setting.

Concise, usable model outputs from preference tuning can boost RL performance. Applying length control during DPO substantially reduces the model's average generation length, allowing us to trade off some performance for improved conciseness and overall usability. While this reduction in length comes with lower scores on length-sensitive evaluations—particularly math benchmarks such as AIME and MATH500—our internal qualitative assessments ("vibe tests") almost uniformly preferred the shorter, more direct model. We make a conscious decision to prioritize usability.

Surprisingly, despite the lower benchmark performance at the DPO stage, length control yielded a more

---

[44]Initial experiments with GPT-judged data showed similar trends.

	LitQA2			SimpleQA		
	No tools	ASC	Δ	No tools	SBT	Δ
Olmo 3 Instruct 7B	24.4	38.2	13.8	3.3	79.2	75.9
Qwen 3 8B (w/o reasoning)	34.7	39.6	4.9	2.0	79.0	77.0
Qwen 3 VL 8B Instruct	34.7	30.7	-4.0	9.3	90.3	81.0
Qwen 2.5 7B	36.0	29.8	-6.2	3.3	78.0	74.7

Table 28 Comparison of agents' performance with and without access to tools on LitQA2 and SimpleQA. ASC refers to Asta Scientific Corpus tools and SBT refers to search and browsing tools.

Name	Subset of Olmo 3 Instruct Benchmarks									
	Avg.	MMLU	BBH	GPQA	AGI	MATH	CHE	LCB	IFEval	AE2
Development 7B SFT	51.9	67.6	47.7	30.2	62.0	65.5	69.3	17.9	83.2	23.8
Olmo 2 preference data	55.5	69.4	55.6	33.7	63.6	71.3	73.7	12.7	**84.5**	35.2
GPT UltraF. pipeline	55.4	67.6	51.2	31.5	61.8	72.2	71.5	14.7	80.8	47.5
+ Sample weak models	56.3	68.4	50.4	33.9	63.8	71.6	74.3	18.2	81.9	44.4
+ Min score rejected	57.4	68.5	53.6	34.4	64.2	72.6	**75.2**	19.1	82.3	47.0
Delta learning only	57.6	68.7	49.5	**35.5**	**64.6**	79.1	73.9	**22.0**	78.6	46.1
Delta learning + GPT	**60.4**	**69.4**	**66.9**	34.6	64.3	**80.0**	74.1	21.1	83.0	**49.8**

Table 29 Development results on comparing different preference signals: preference pairs created with the delta learning heuristic (chosen = large model response, rejected = smaller model response) and pairs created with our delta-aware LLM-judge pipeline yield a different spread of gains, suggesting that they provide different preference signals. These signals are complementary; combining them both yields the largest average gain. Our final OLMO 3 INSTRUCT preference data greatly outperforms our previous OLMO 2 preference data.

performant model post RL. We conjecture that this arises from the RL training context: with a fixed context window (8K), a shorter model may be "more intelligent per token," allowing it to leverage the available budget more effectively during optimization. Thus, what initially appeared to be a tradeoff between usability and performance ultimately produced improvements in both. Moreover, we found that RL training progresses more reliably when initialized from the length-controlled DPO policy. Across most benchmarks, performance improves more steadily compared to RL runs starting from a higher-scoring but unconstrained DPO checkpoint, which tends to show earlier signs of instability or degradation. This further supports the role of concise preference-tuned models as advantageous starting points for RL.

Need for tools We assessed how much of OLMO 3 INSTRUCT's performance on LitQA2 and SimpleQA can actually be attributed to tool use by measuring the delta of the model performance on the benchmarks between answering the questions only from parametric memory (No tools) and doing so using tools. Table 28 shows these deltas in comparison to those from three Qwen models. All models benefit a lot from tool use on SimpleQA. However, Qwen models, unlike OLMO 3 INSTRUCT 7B, mostly seem to rely on parametric knowledge for LitQA2, with two of the models even losing performance when provided with tools.

6 Olmo 3 RL-Zero

RL has become a key part of recent LLM pipelines in part due to Deepseek R1-Zero (Guo et al., 2025) which notably leverage RL training on top of a base model to bootstrap complex reasoning behavior (Marjanović et al., 2025). This has made RLVR finetuning from a base model the standard large-scale benchmark for RL algorithms (Liu et al., 2025a; Yu et al., 2025; Luo et al., 2025b). To date, all leading open RLVR benchmarks and algorithms train on top of open-weights models that do not reveal their pretraining or mid-training data (Chu et al., 2025; Yang et al., 2025a). This limits the ability for the community to study the role of pretraining data on RLVR performance. It can lead to a myriad of issues with benchmark evaluations being contaminated e.g. mid-training data containing the evaluation which makes spurious rewards as effective as true reward (Shao et al., 2025; Wu et al., 2025c) or improvements from fixing prompt templates outweighing the improvements from RL (Liu et al., 2025b).

We therefore release a fully open dataset Dolci RL-Zero, an algorithmic RL zero setup for Olmo 3, and open-source OlmoRL code to enable clear benchmarking for the ecosystem. We perform RLVR from Olmo 3 Base over four benchmarking domains to create the Olmo 3 RL-Zero family: math, code, precise instruction following (IF) and a mix of all three. In all cases, we further decontaminate Dolci RL-Zero from pretraining and midtraining data to guarantee our setup carefully studies the effect of RLVR without data leakage confounding our conclusions.

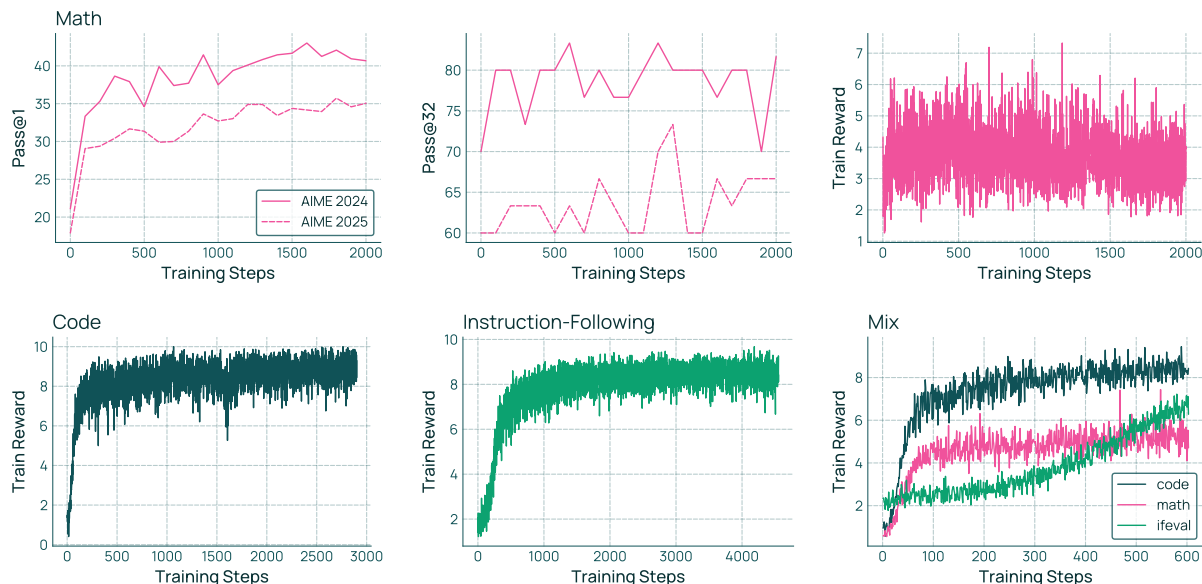## 6.1 Reinforcement Learning From Base with Dolci RL-Zero

Data   We create Dolci RL-Zero, an effective RL-zero training dataset. For Math, we do an aggressive filtering of DAPO math (Yu et al., 2025) and Klear-Reasoner Math (Su et al., 2025c). We de-deduplicate DAPO and remove all non-English examples. As Klear-Reasoner is a much larger dataset, we further group questions with semantic clustering and select one representative question per cluster. We further decontaminate against both pretraining and evaluation data following subsubsection 4.2.1 and perform offline filtering for prompts fully solved in 8 out of 8 sample completions by the final base model. This results in a dataset of 13.3k math prompts. Data for code and instruction-following is subsampled from Dolci Think RL.

Prompt and Eval. Template   Confirming the findings of Liu et al. (2025b), we find that "simple" prompt templates greatly outperform standard post-trained templates (e.g. `<think></think>`) when training from a purely mid-trained model, as Dolma 3 Dolmino Mix excluded most special formatting. We develop a simple custom prompt for each domain, using the zero-shot pass@k performance as our metric. We end up with a prompt similar to Yu et al. (2025), shown in Figure 33. We furthermore "clean" all our evaluation prompts to remove special formatting (i.e. `\boxed{}`) to make evaluations prompts more similar to training.
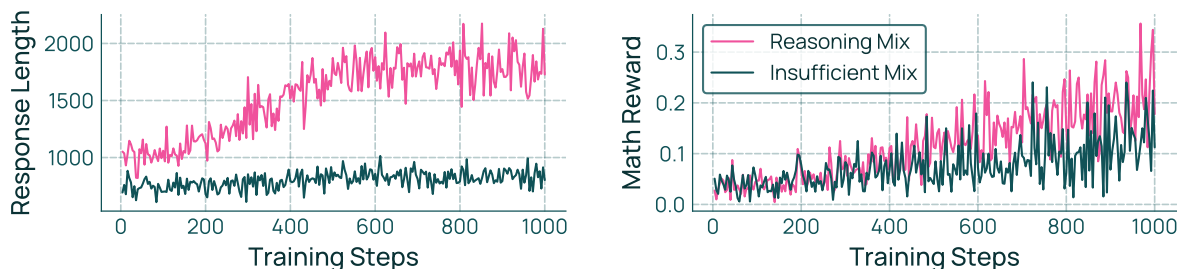
RL Algorithm   We follow Section § 4.4.1 in all RL details except (i) we train with response length of 16K tokens to better accommodate long chain-of-though reasoning in math and code domains and (ii) we evaluate with a response length of 32K tokens and temperature 1.0 to encourage diversity as we report pass@k. See Table 47 for hyperparameter details.

6.2 Key Findings

Olmo 3 RL-Zero can strongly improve on reasoning   As shown in Figure 24, our base model can greatly improve on train reward across the different domains when leveraging RL on our datasets. To demonstrate OOD improvements, we evaluate our math run on the decontaminated evals AIME 2024 and 2025. We find that Olmo 3 Base drastically improves in the first couple hundred steps of training and then improves steadily but slowly. We also see slight improvement in Pass@32 results, demonstrating that our run maintains diversity and RLVR pushes the model beyond its initial capabilities. Our initial scores and final scores with the 7B model are, notably, close to DAPO (Yu et al., 2025) which leverages the larger Qwen 2.5 32B and trains for an order of magnitude more steps. This demonstrates how Olmo 3 RL-Zero can be a more efficient alternative to existing RLVR experiments.
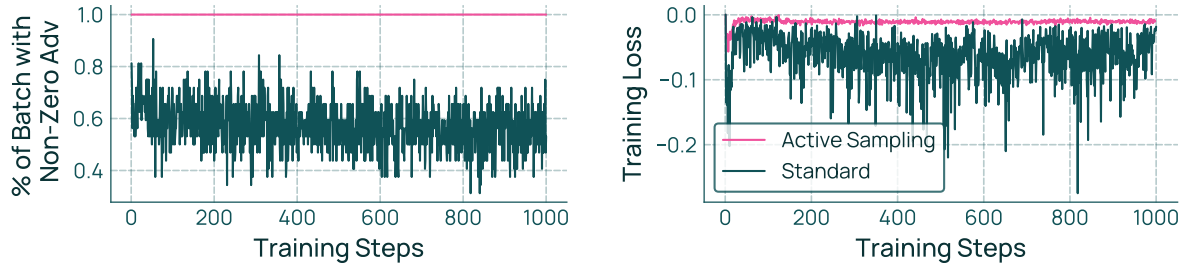
Figure 24 Different domain runs of RL-Zero on OLMO 3 BASE: math, precise instruction-following, code, and a mix of all three. We show the main evaluation for the math domain: AIME 2024 and 2025 with Pass@1, computed as a bootstrapped average over 32 samples, and Pass@32. For all domains, we show reward over training. For Mix, we separate out the individual rewards for each domain.
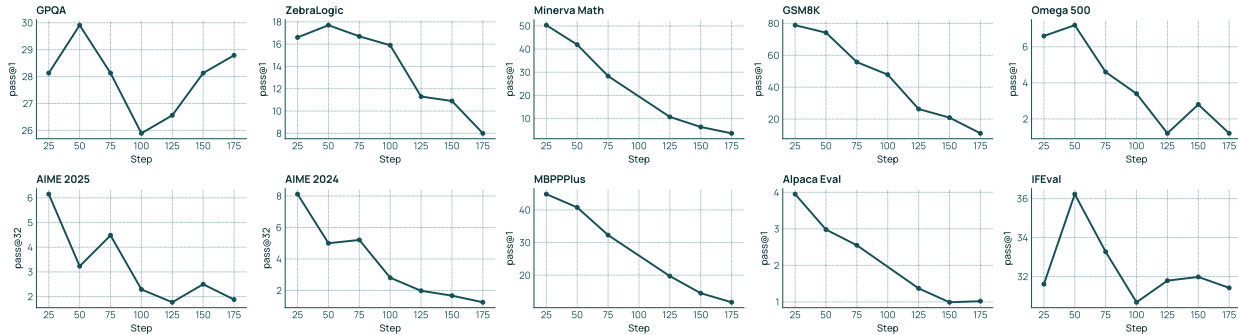

Figure 25 The response length and math reward over RL training for two early midtrained base models. This demonstrates the how base model midtraining can determine whether RL-zero learns longer, complex reasoning and increases response length.

Olmo 3 RL-Zero Mix can benchmark challenges in multi-objective RL Most studies have focused exclusively on RLHF (Stiennon et al., 2020) or single-domain RLVR (Yu et al., 2025; Luo et al., 2025a). Our mix of math, code, and instruction-following is a more challenging RLVR benchmark for models. Figure 24 demonstrates that our general run has improved performance across different domains but each domain is under-optimized compared to the single-domain setup. Future work can leverage this setup to investigate the interactions between domains in multi-objective RLVR.

Olmo 3 RL-Zero can benchmark reasoning data mixes in midtraining Midtraining and OLMO 3 RL-ZERO offers a chance to ablate specific data sources, unlike the large scale effort behind OLMO 3 THINK. We leveraged RL-Zero to evaluate midtraining data mixes for their ability to develop downstream reasoning with RL. For example, we compare two early models in Figure 25. As evidenced by the stagnant response length, the model with insufficient reasoning data did not learn to backtrack, verify answers, and other cognitive skills (Gandhi et al., 2025). OLMO 3 RL-ZERO can therefore serve as a testbed for downstream performance of alternative midtraining approaches and improvements over DOLMA 3 DOLMINO MIX.

Figure 26 Active Sampling maintains a full batch of non-zero advantage samples by continuously pulling prompt-completions from the result queue after filtering. We plot the percentage of batch with non-zero advantage as well as the train loss for an RL-Zero Math run with and without active sampling.

Figure 27 RL training on OLMO 3 BASE using random, signal-free rewards produces no performance gains, suggesting successful decontamination of training data.

Active Sampling stabilizes training   OLMO 3 RL-ZERO also offers a simpler test bed for ablating RL algorithm and infrastructure decisions. We ablate active sampling, our novel method for continuously re-sampling prompts after filtering for non-zero advantage (see OLMORL Infra). Running on our Math domain, Figure 26 shows that active sampling does indeed maintain consistent full batch of completions with non-zero advantage. These consistent batch sizes have an interesting effect of training stability and we see greatly reduced loss variance.

Eval decontamination is verified via Spurious Rewards   Recent RLVR benchmarks have shown substantial improvements from training with spurious rewards that are not correlated with model utility. This can suggest that the RLVR task may have been *contaminated* i.e. the model exposed to evaluation data during pretraining or midtraining. RLVR with a spurious reward can elicit this memorized knowledge, differentiating it from genuine learning of reasoning capabilities (Shao et al., 2025). To verify that OLMO 3 RL-ZERO evaluation is not contaminated, we conduct a negative control experiment by training OLMO 3 BASE with spurious rewards on a standard RLVR dataset. Specifically, we assign random binary rewards to model generations independent of correctness, following the protocol in Shao et al. (2025). We use the decontaminated OpenReasoner Zero (Hu et al., 2025). If our pretraining or midtraining data contained significant overlap with our evaluation sets, we would expect spurious reward training to elicit these memorized solutions and improve benchmark performance.

As shown in Figure 27, training with random rewards does not improve performance on any of our benchmark evals. Performance either remains flat with random fluctuations or degrades, which is consistent with the model learning arbitrary patterns unrelated to the task. This negative result is evidence that our data decontamination successfully removed overlaps between our base model pipeline and RLVR evaluation data.

References

M. Abdin, J. Aneja, H. S. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, J. R. Lee, Y. T. Lee, Y. Li, W. Liu, C. C. T. Mendes, A. Nguyen, E. Price, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, X. Wang, R. Ward, Y. Wu, D. Yu, C. Zhang, and Y. Zhang. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

R. Ackerman and V. A. Thompson. Meta-reasoning: Monitoring and control of thinking and reasoning. *Trends in Cognitive Sciences*, 21(8):607–617, 2017. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2017.05.004. URL https://www.sciencedirect.com/science/article/pii/S1364661317301055.

B. Adler, N. Agarwal, A. Aithal, D. H. Anh, P. Bhattacharya, A. Brundyn, J. Casper, B. Catanzaro, S. Clay, J. Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

P. Aggarwal and S. Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.04697.

W. U. Ahmad, S. Narenthiran, S. Majumdar, A. Ficek, S. Jain, J. Huang, V. Noroozi, and B. Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025. URL https://arxiv.org/abs/2504.01943.

S. N. Akter, S. Prabhumoye, J. Kamalu, S. Satheesh, E. Nyberg, M. Patwary, M. Shoeybi, and B. Catanzaro. Mind: Math informed synthetic dialogues for pretraining llms, 2024. URL https://arxiv.org/abs/2410.12881.

L. B. Allal, A. Lozhkov, E. Bakouch, G. M. Blázquez, G. Penedo, L. Tunstall, A. Marafioti, H. Kydlíček, A. P. Lajarín, V. Srivastav, J. Lochner, C. Fahlgren, X.-S. Nguyen, C. Fourrier, B. Burtenshaw, H. Larcher, H. Zhao, C. Zakka, M. Morlon, C. Raffel, L. von Werra, and T. Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

C. An, Z. Xie, X. Li, L. Li, J. Zhang, S. Gong, M. Zhong, J. Xu, X. Qiu, M. Wang, and L. Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Anthropic. System card: Claude opus 4 & claude sonnet 4. Technical report, Anthropic, 2025. Accessed: 2025-10-07.

Apertus Team. Apertus: Democratizing Open and Compliant LLMs for Global Language Environments. https://huggingface.co/swiss-ai/Apertus-70B-2509, 2025.

A. Asai, J. He, R. Shao, W. Shi, A. Singh, J. C. Chang, K. Lo, L. Soldaini, S. Feldman, M. D'Arcy, D. Wadden, M. Latzke, M. Tian, P. Ji, S. Liu, H. Tong, B. Wu, Y. Xiong, L. S. Zettlemoyer, G. Neubig, D. Weld, D. Downey, W. tau Yih, P. W. Koh, and H. Hajishirzi. Openscholar: Synthesizing scientific literature with retrieval-augmented lms. *ArXiv*, abs/2411.14199, 2024. URL https://api.semanticscholar.org/CorpusID:274166189.

G. Attardi. Wikiextractor. https://github.com/attardi/wikiextractor, 2015.

J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

M. G. Azar, M. Rowland, B. Piot, D. Guo, D. Calandriello, M. Valko, and R. Munos. A general theoretical paradigm to understand learning from human preferences, 2023. URL https://arxiv.org/abs/2310.12036.

Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck. Llemma: An open language model for mathematics, 2023.

Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li. LongBench: A bilingual, multitask benchmark for long context understanding. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Y. Bai, S. Tu, J. Zhang, H. Peng, X. Wang, X. Lv, S. Cao, J. Xu, L. Hou, Y. Dong, J. Tang, and J. Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, Vienna, Austria, July 2025. Association for Computational

Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.183. URL https://aclanthology.org/2025.acl-long.183/.

E. Bakouch, L. Ben Allal, A. Lozhkov, N. Tazi, L. Tunstall, C. M. Patiño, E. Beeching, A. Roucher, A. J. Reedi, Q. Gallouédec, K. Rasul, N. Habib, C. Fourrier, H. Kydlicek, G. Penedo, H. Larcher, M. Morlon, V. Srivastav, J. Lochner, X.-S. Nguyen, C. Raffel, L. von Werra, and T. Wolf. SmolLM3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3, 2025.

M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

A. Bercovich, I. Levy, I. Golan, M. Dabbah, R. El-Yaniv, O. Puny, I. Galil, Z. Moshe, T. Ronen, N. Nabwani, I. Shahaf, O. Tropp, E. Karpas, R. Zilberstein, J. Zeng, S. Singhal, A. Bukharin, Y. Zhang, T. Konuk, G. Shen, A. S. Mahabaleshwarkar, B. Kartal, Y. Suhara, O. Delalleau, Z. Chen, Z. Wang, D. Mosallanezhad, A. Renduchintala, H. Qian, D. Rekesh, F. Jia, S. Majumdar, V. Noroozi, W. U. Ahmad, S. Narenthiran, A. Ficek, M. Samadi, J. Huang, S. Jain, I. Gitman, I. Moshkov, W. Du, S. Toshniwal, G. Armstrong, B. Kisacanin, M. Novikov, D. Gitman, E. Bakhturina, J. P. Scowcroft, J. Kamalu, D. Su, K. Kong, M. Kliegl, R. Karimi, Y. Lin, S. Satheesh, J. Parmar, P. Gundecha, B. Norick, J. Jennings, S. Prabhumoye, S. N. Akter, M. Patwary, A. Khattar, D. Narayanan, R. Waleffe, J. Zhang, B.-Y. Su, G. Huang, T. Kong, P. Chadha, S. Jain, C. Harvey, E. Segal, J. Huang, S. Kashirsky, R. McQueen, I. Putterman, G. Lam, A. Venkatesan, S. Wu, V. Nguyen, M. Kilaru, A. Wang, A. Warno, A. Somasamudramath, S. Bhaskar, M. Dong, N. Assaf, S. Mor, O. U. Argov, S. Junkin, O. Romanenko, P. Larroy, M. Katariya, M. Rovinelli, V. Balas, N. Edelman, A. Bhiwandiwalla, M. Subramaniam, S. Ithape, K. Ramamoorthy, Y. Wu, S. V. Velury, O. Almog, J. Daw, D. Fridman, E. Galinkin, M. Evans, K. Luna, L. Derczynski, N. Pope, E. Long, S. Schneider, G. Siman, T. Grzegorzek, P. Ribalta, M. Katariya, J. Conway, T. Saar, A. Guan, K. Pawelec, S. Prayaga, O. Kuchaiev, B. Ginsburg, O. Olabiyi, K. Briski, J. Cohen, B. Catanzaro, J. Alben, Y. Geifman, E. Chung, and C. Alexiuk. Llama-nemotron: Efficient reasoning models, 2025. URL https://arxiv.org/abs/2505.00949.

R. Berry, M. Jordan, and L. Soldaini. Decon: Decontamination tools. https://github.com/allenai/decon, 2025. Accessed: 2025-11-14.

J. Bevendorff, B. Stein, M. Hagen, and M. Potthast. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In L. Azzopardi, A. Hanbury, G. Pasi, and B. Piwowarski, editors, *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York, Mar. 2018. Springer.

A. Bhagia, J. Liu, A. Wettig, D. Heineman, O. Tafjord, A. H. Jha, L. Soldaini, N. A. Smith, D. Groeneveld, P. W. Koh, J. Dodge, and H. Hajishirzi. Establishing task scaling laws via compute-efficient model ladders, 2024. URL https://arxiv.org/abs/2412.04403.

Y. Bisk, R. Zellers, R. Le bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

S. Bordt, S. Srinivas, V. Boreiko, and U. von Luxburg. How much can we forget about data contamination? *ArXiv*, abs/2410.03249, 2024. URL https://api.semanticscholar.org/CorpusID:273163321.

J. Bragg, M. D'Arcy, N. Balepur, D. Bareket, B. Dalvi, S. Feldman, D. Haddad, J. D. Hwang, P. Jansen, V. Kishore, et al. Astabench: Rigorous benchmarking of ai agents with a scientific research suite. *arXiv preprint arXiv:2510.21652*, 2025.

F. Brahman, S. Kumar, V. Balachandran, P. Dasigi, V. Pyatkin, A. Ravichander, S. Wiegreffe, N. Dziri, K. Chandu, J. Hessel, et al. The art of saying no: Contextual noncompliance in language models. *arXiv preprint arXiv:2407.12043*, 2024.

Z. Cai, S. Shabihi, B. An, Z. Che, B. R. Bartoldson, B. Kailkhura, T. Goldstein, and F. Huang. Aegisllm: Scaling agentic systems for self-reflective defense in llm security. *arXiv preprint arXiv:2504.20965*, 2025. Preprint.

F. Callaway, B. {van Opheusden}, S. Gul, P. Das, P. Krueger, T. Griffiths, and F. Lieder. Rational use of cognitive resources in human planning. *Nature Human Behaviour*, 6(8):1112–1125, Aug. 2022. ISSN 2397-3374. doi: 10.1038/s41562-022-01332-8. Publisher Copyright: © 2022, The Author(s), under exclusive licence to Springer Nature Limited.

F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

A. Chatterji, T. Cunningham, D. Deming, Z. Hitzig, C. Ong, C. Y. Shan, and K. Wadman. How people use ChatGPT. Technical Report w34255, National Bureau of Economic Research, Cambridge, MA, Sept. 2025.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. 2021.

M. F. Chen, T. Murray, D. Heineman, M. Jordan, H. Hajishirzi, C. Ré, L. Soldaini, and K. Lo. Olmix: Efficient mixture recomputation for evolving lm datasets, 2025a.

S. Chen, S. Wong, L. Chen, and Y. Tian. Extending context window of large language models via positional interpolation, 2023. URL https://arxiv.org/abs/2306.15595.

Y. Chen, Z. Yang, Z. Liu, C. Lee, P. Xu, M. Shoeybi, B. Catanzaro, and W. Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning. *arXiv preprint arXiv:2505.16400*, 2025b.

Z. Cheng, S. Hao, T. Liu, F. Zhou, Y. Xie, F. Yao, Y. Bian, Y. Zhuang, N. Dey, Y. Zha, Y. Gu, K. Zhou, Y. Wang, Y. Li, R. Fan, J. She, C. Gao, A. Saparov, H. Li, T. W. Killian, M. Yurochkin, Z. Liu, E. P. Xing, and Z. Hu. Revisiting reinforcement learning for llm reasoning from a cross-domain perspective, 2025. URL https://arxiv.org/abs/2506.14965.

W. Chu, X. Xie, J. Yu, J. Wang, A. Phanishayee, C. Tang, Y. Hao, J. Huang, M. Ozdal, J. Wang, V. Goswami, N. Goyal, A. Kadian, A. Gu, C. Cai, F. Tian, X. Wang, M. Si, P. Balaji, C.-H. Chu, and J. Park. Scaling llama 3 training with efficient parallelism strategies. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, page 1703–1716, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712616. doi: 10.1145/3695053.3731410. URL https://doi.org/10.1145/3695053.3731410.

C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *CoRR*, arXiv:1803.05457, 2018.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Common Crawl Foundation. Common Crawl Dataset. https://commoncrawl.org/. Accessed: December 31, 2024.

G. Cui, L. Yuan, N. Ding, G. Yao, B. He, W. Zhu, Y. Ni, G. Xie, R. Xie, Y. Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*.

G. Cui, L. Yuan, N. Ding, G. Yao, W. Zhu, Y. Ni, G. Xie, Z. Liu, and M. Sun. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*, 2023.

P. Dasigi, K. Lo, I. Beltagy, A. Cohan, N. A. Smith, and M. Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

DeepSeek-AI. DeepSeek-V3.1 release. https://api-docs.deepseek.com/news/news250821, 2025. Accessed: 2025-11-10.

DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang,

R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

S. Diao, Y. Yang, Y. Fu, X. Dong, D. Su, M. Kliegl, Z. Chen, P. Belcak, Y. Suhara, H. Yin, M. Patwary, Yingyan, Lin, J. Kautz, and P. Molchanov. Climb: Clustering-based iterative data mixture bootstrapping for language model pre-training, 2025. URL https://arxiv.org/abs/2504.13161.

H. Ding, Z. Wang, G. Paolini, V. Kumar, A. Deoras, D. Roth, and S. Soatto. Fewer truncations improve language modeling, 2024. URL https://arxiv.org/abs/2404.10830.

N. Ding, Y. Chen, B. Xu, Y. Qin, S. Hu, Z. Liu, M. Sun, and B. Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, 2023.

D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246.

Y. Dubois, B. Galambosi, P. Liang, and T. B. Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

A. Fan, Y. Jernite, E. Perez, D. Grangier, J. Weston, and M. Auli. Eli5: Long form question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3558–3567, 2019.

A. Fang, H. Pouransari, M. Jordan, A. Toshev, V. Shankar, L. Schmidt, and T. Gunter. Datasets, documents, and repetitions: The practicalities of unequal data quality, 2025a. URL https://arxiv.org/abs/2503.07879.

L. Fang, Y. Wang, Z. Liu, C. Zhang, S. Jegelka, J. Gao, B. Ding, and Y. Wang. What is wrong with perplexity for long-context language modeling?, 2025b. URL https://arxiv.org/abs/2410.23771.

S. Fleming and N. Daw. Self-evaluation of decision-making: A general bayesian framework for metacognitive computation. *Psychological Review*, 124(1):91–114, 2017. doi: 10.1037/rev0000045.

K. Fujii, Y. Tajima, S. Mizuki, H. Shimada, T. Shiotani, K. Saito, M. Ohi, M. Kawamura, T. Nakamura, T. Okamoto, et al. Rewriting pre-training data boosts llm performance in math and code. *arXiv preprint arXiv:2505.02881*, 2025.

K. Gandhi, A. Chakravarthy, A. Singh, N. Lile, and N. D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.

L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

T. Gao, A. Wettig, H. Yen, and D. Chen. How to train long-context language models (effectively). In *ACL*, 2025.

Gemma 3 Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Gemma Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

S. Geng, H. Ivison, C.-L. Li, M. Sap, J. Li, R. Krishna, and P. W. Koh. The delta learning hypothesis: Preference tuning on weak data can yield strong gains. *arXiv preprint arXiv:2507.06187*, 2025.

GLM-4.5 Team, A. Zeng, X. Lv, Q. Zheng, Z. Hou, B. Chen, C. Xie, C. Wang, D. Yin, H. Zeng, J. Zhang, K. Wang, L. Zhong, M. Liu, R. Lu, S. Cao, X. Zhang, X. Huang, Y. Wei, Y. Cheng, Y. An, Y. Niu, Y. Wen, Y. Bai, Z. Du, Z. Wang, Z. Zhu, B. Zhang, B. Wen, B. Wu, B. Xu, C. Huang, C. Zhao, C. Cai, C. Yu, C. Li, C. Ge, C. Huang, C. Zhang, C. Xu, C. Zhu, C. Li, C. Yin, D. Lin, D. Yang, D. Jiang, D. Ai, E. Zhu, F. Wang, G. Pan, G. Wang, H. Sun, H. Li, H. Li, H. Hu, H. Zhang, H. Peng, H. Tai, H. Zhang, H. Wang, H. Yang, H. Liu, H. Zhao, H. Liu,

H. Yan, H. Liu, H. Chen, J. Li, J. Zhao, J. Ren, J. Jiao, J. Zhao, J. Yan, J. Wang, J. Gui, J. Zhao, J. Liu, J. Li, J. Li, J. Lu, J. Wang, J. Yuan, J. Li, J. Du, J. Du, J. Liu, J. Zhi, J. Gao, K. Wang, L. Yang, L. Xu, L. Fan, L. Wu, L. Ding, L. Wang, M. Zhang, M. Li, M. Xu, M. Zhao, M. Zhai, P. Du, Q. Dong, S. Lei, S. Tu, S. Yang, S. Lu, S. Li, S. Li, Shuang-Li, S. Yang, S. Yi, T. Yu, W. Tian, W. Wang, W. Yu, W. L. Tam, W. Liang, W. Liu, X. Wang, X. Jia, X. Gu, X. Ling, X. Wang, X. Fan, X. Pan, X. Zhang, X. Zhang, X. Fu, X. Zhang, Y. Xu, Y. Wu, Y. Lu, Y. Wang, Y. Zhou, Y. Pan, Y. Zhang, Y. Wang, Y. Li, Y. Su, Y. Geng, Y. Zhu, Y. Yang, Y. Li, Y. Wu, Y. Li, Y. Liu, Y. Wang, Y. Li, Y. Zhang, Z. Liu, Z. Yang, Z. Zhou, Z. Qiao, Z. Feng, Z. Liu, Z. Zhang, Z. Wang, Z. Yao, Z. Wang, Z. Liu, Z. Chai, Z. Li, Z. Zhao, W. Chen, J. Zhai, B. Xu, M. Huang, H. Wang, J. Li, Y. Dong, and J. Tang. GLM-4.5: Agentic, Reasoning, and Coding (ARC) Foundation Models, 2025. URL https://arxiv.org/abs/2508.06471.

C. Goddard. Extending AFM-4.5B to 64k context length. https://www.arcee.ai/blog/extending-afm-4-5b-to-64k-context-length, June 2025. Accessed: 2025-11-10.

C. Goddard, S. Siriwardhana, M. Ehghaghi, L. Meyers, V. Karpukhin, B. Benedict, M. McQuade, and J. Solawetz. Arcee's MergeKit: A toolkit for merging large language models. In F. Dernoncourt, D. Preoţiuc-Pietro, and A. Shimorina, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 477–485, Miami, Florida, US, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-industry.36. URL https://aclanthology.org/2024.emnlp-industry.36.

A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu,

66

M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

T. L. Griffiths, F. Callaway, M. B. Chang, E. Grant, P. M. Krueger, and F. Lieder. Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2019.01.005. URL `https://www.sciencedirect.com/science/article/pii/S2352154618302122`. Artificial Intelligence.

A. Gu, B. Rozière, H. Leather, A. Solar-Lezama, G. Synnaeve, and S. I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024a.

Y. Gu, O. Tafjord, B. Kuehl, D. Haddad, J. Dodge, and H. Hajishirzi. Olmes: A standard for language model evaluations. *ArXiv*, abs/2406.08446, 2024b. URL `https://api.semanticscholar.org/CorpusID:270391754`.

E. Guha, R. Marten, S. Keh, N. Raoof, G. Smyrnis, H. Bansal, M. Nezhurina, J. Mercat, T. Vu, Z. Sprague, A. Suvarna, B. Feuer, L. Chen, Z. Khan, E. Frankel, S. Grover, C. Choi, N. Muennighoff, S. Su, W. Zhao, J. Yang, S. Pimpalgaonkar, K. Sharma, C. C.-J. Ji, Y. Deng, S. Pratt, V. Ramanujan, J. Saad-Falcon, J. Li, A. Dave, A. Albalak, K. Arora, B. Wulfe, C. Hegde, G. Durrett, S. Oh, M. Bansal, S. Gabriel, A. Grover, K.-W. Chang, V. Shankar, A. Gokaslan, M. A. Merrill, T. Hashimoto, Y. Choi, J. Jitsev, R. Heckel, M. Sathiamoorthy, A. G. Dimakis, and L. Schmidt. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025a. URL `https://arxiv.org/abs/2506.04178`.

E. Guha, R. Marten, S. Keh, N. Raoof, G. Smyrnis, H. Bansal, M. Nezhurina, J. Mercat, T. Vu, Z. Sprague, A. Suvarna, B. Feuer, L. Chen, Z. Khan, E. Frankel, S. Grover, C. Choi, N. Muennighoff, S. Su, W. Zhao, J. Yang, S. Pimpalgaonkar, K. Sharma, C. C.-J. Ji, Y. Deng, S. Pratt, V. Ramanujan, J. Saad-Falcon, J. Li, A. Dave, A. Albalak, K. Arora, B. Wulfe, C. Hegde, G. Durrett, S. Oh, M. Bansal, S. Gabriel, A. Grover, K.-W. Chang, V. Shankar, A. Gokaslan, M. A. Merrill, T. Hashimoto, Y. Choi, J. Jitsev, R. Heckel, M. Sathiamoorthy, A. G. Dimakis, and L. Schmidt. Openthoughts: Data recipes for reasoning models, 2025b. URL `https://arxiv.org/abs/2506.04178`.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

D. Hall, C. Chou, A. Garg, N. Ravi, N. Liu, H. Shandilya, A. Ahmed, P. Liang, R. Kuditipudi, J38, T. Lee, R. Power, K. Salahi, W. Held, J. Wang, chiheem, J. Niklaus, Y. Mai, dependabot[bot], I. Zhou, K. X. Li, S. Yang, S. Karamcheti, R. Williams, C. Zhou, A. Ramaswami, whenwen, S. Kotha, G. Miguel, and C. Xu. marin-community/marin. https://github.com/marin-community/marin, nov 14 2025. URL `https://github.com/marin-community/marin`.

S. Han, K. Rao, A. Ettinger, L. Jiang, B. Y. Lin, N. Lambert, Y. Choi, and N. Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *arXiv preprint arXiv:2406.18495*, 2024.

T. Hartvigsen, S. Gabriel, H. Palangi, M. Sap, D. Ray, and E. Kamar. Toxigen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection. pages 3309–3326, 01 2022. doi: 10.18653/v1/2022.acl-long.234.

G. Haupt. Hierarchical thinking: a cognitive tool for guiding coherent decision making in design problem solving. *International Journal of Technology and Design Education*, 28(1):207–237, 2018. ISSN 1573-1804. doi: 10.1007/s10798-016-9381-0. URL `https://doi.org/10.1007/s10798-016-9381-0`.

D. Heineman, V. Hofmann, I. Magnusson, Y. Gu, N. A. Smith, H. Hajishirzi, K. Lo, and J. Dodge. Signal and noise: A framework for reducing uncertainty in language model evaluation, 2025. URL `https://arxiv.org/abs/2508.13144`.

D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021a.

D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021c.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

J. Hu, Y. Zhang, Q. Han, D. Jiang, X. Zhang, and H.-Y. Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

Y. Huang, L. Sun, H. Wang, S. Wu, Q. Zhang, Y. Li, C. Gao, Y. Huang, W. Lyu, Y. Zhang, et al. Trustllm: Trustworthiness in large language models. *arXiv preprint arXiv:2401.05561*, 2024a.

Y. Huang, J. Zhang, Z. Shan, and J. He. Compression represents intelligence linearly. *arXiv preprint arXiv:2404.09937*, 2024b.

N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

L. Jiang, K. Rao, S. Han, A. Ettinger, F. Brahman, S. Kumar, N. Mireshghallah, X. Lu, M. Sap, Y. Choi, and N. Dziri. Wildteaming at scale: From in-the-wild jailbreaks to (adversarially) safer language models. *arXiv preprint arXiv:2406.18510*, 2024. URL https://arxiv.org/abs/2406.18510.

Z. Jiang, M. Y. R. Yang, M. Tsirlin, R. Tang, and J. Lin. Less is more: Parameter-free text classification with gzip, 2022. URL https://arxiv.org/abs/2212.09410.

D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

J. M. Joyce. Causal reasoning and backtracking. *Philosophical Studies*, 147(1):139–154, 2009. doi: 10.1007/s11098-009-9454-y.

F. Kaiyom, A. Ahmed, Y. Mai, K. Klyman, R. Bommasani, and P. Liang. HELM safety: Towards standardized safety evaluations of language models, 8 Nov. 2024. URL https://crfm.stanford.edu/2024/11/08/helm-safety.html.

P. Kargupta, S. S. Li, H. Wang, J. Lee, S. Chen, O. Ahia, D. Light, T. L. Griffiths, M. Kleiman-Weiner, J. Han, A. Celikyilmaz, and Y. Tsvetkov. Cognitive foundations for reasoning and their manifestation in llms. *arXiv*, 2025.

K. Kavukcuoğlu and G. DeepMind. Gemini 2.5: Our most intelligent ai model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/, Mar. 2025. Accessed: 2025-10-07.

S. Kim, S. Bae, J. Shin, S. Kang, D. Kwak, K. Yoo, and M. Seo. Aligning large language models through synthetic feedback. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13677–13700, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.844. URL https://aclanthology.org/2023.emnlp-main.844/.

Kimi Team, Y. Bai, Y. Bao, G. Chen, J. Chen, N. Chen, R. Chen, Y. Chen, Y. Chen, Y. Chen, Z. Chen, J. Cui, H. Ding, M. Dong, A. Du, C. Du, D. Du, Y. Du, Y. Fan, Y. Feng, K. Fu, B. Gao, H. Gao, P. Gao, T. Gao, X. Gu, L. Guan, H. Guo, J. Guo, H. Hu, X. Hao, T. He, W. He, W. He, C. Hong, Y. Hu, Z. Hu, W. Huang, Z. Huang, Z. Huang, T. Jiang, Z. Jiang, X. Jin, Y. Kang, G. Lai, C. Li, F. Li, H. Li, M. Li, W. Li, Y. Li, Y. Li, Z. Li, Z. Li, H. Lin, X. Lin, Z. Lin, C. Liu, C. Liu, H. Liu, J. Liu, J. Liu, L. Liu, S. Liu, T. Y. Liu, T. Liu, W. Liu, Y. Liu, Y. Liu, Y. Liu, Y. Liu, Z. Liu, E. Lu, L. Lu, S. Ma, X. Ma, Y. Ma, S. Mao, J. Mei, X. Men, Y. Miao, S. Pan, Y. Peng, R. Qin, B. Qu, Z. Shang, L. Shi, S. Shi, F. Song, J. Su, Z. Su, X. Sun, F. Sung, H. Tang, J. Tao, Q. Teng, C. Wang, D. Wang, F. Wang, H. Wang, J. Wang, J. Wang, J. Wang, S. Wang, S. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Z. Wang, Z. Wang, Z. Wang, C. Wei, Q. Wei, W. Wu, X. Wu, Y. Wu, C. Xiao, X. Xie, W. Xiong, B. Xu, J. Xu, J. Xu, L. H. Xu, L. Xu, S. Xu, W. Xu, X. Xu, Y. Xu, Z. Xu, J. Yan, Y. Yan, X. Yang, Y. Yang, Z. Yang, Z. Yang, Z. Yang, H. Yao, X. Yao, W. Ye, Z. Ye, B. Yin, L. Yu, E. Yuan, H. Yuan, M. Yuan, H. Zhan, D. Zhang, H. Zhang, W. Zhang, X. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang,

Z. Zhang, H. Zhao, Y. Zhao, H. Zheng, S. Zheng, J. Zhou, X. Zhou, Z. Zhou, Z. Zhu, W. Zhuang, and X. Zu. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

A. Köpf, Y. Kilcher, D. von Rütte, S. Anagnostidis, Z. R. Tam, K. Stevens, A. Barhoum, D. Nguyen, O. Stanley, R. Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

S. Kudugunta, I. Caswell, B. Zhang, X. Garcia, C. A. Choquette-Choo, K. Lee, D. Xin, A. Kusupati, R. Stella, A. Bapna, and O. Firat. Madlad-400: A multilingual and document-level large audited dataset, 2023. URL https://arxiv.org/abs/2309.04662.

T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-T. Yih, D. Fried, S. Wang, and T. Yu. Ds-1000: A natural and reliable benchmark for data science code generation. *ArXiv*, abs/2211.11501, 2022.

N. Lambert. *Reinforcement Learning from Human Feedback*. Online, 2025. URL https://rlhfbook.com.

N. Lambert, T. K. Gilbert, and T. Zick. Entangled preferences: The history and risks of reinforcement learning and human feedback. *arXiv preprint arXiv:2310.13595*, 2023.

N. Lambert, J. D. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, V. Graf, J. D. Hwang, J. Yang, R. L. Bras, O. Tafjord, C. Wilhelm, L. Soldaini, N. A. Smith, Y. Wang, P. Dasigi, and H. Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. 2024. URL https://api.semanticscholar.org/CorpusID:274192505.

J. M. Laurent, J. D. Janizek, M. Ruzo, M. M. Hinks, M. J. Hammerling, S. Narayanan, M. Ponnapati, A. D. White, and S. G. Rodriques. Lab-bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024.

K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini. Deduplicating training data makes language models better, 2022. URL https://arxiv.org/abs/2107.06499.

A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, S. Garg, R. Xin, N. Muennighoff, R. Heckel, J. Mercat, M. Chen, S. Gururangan, M. Wortsman, A. Albalak, Y. Bitton, M. Nezhurina, A. Abbas, C.-Y. Hsieh, D. Ghosh, J. Gardner, M. Kilian, H. Zhang, R. Shao, S. Pratt, S. Sanyal, G. Ilharco, G. Daras, K. Marathe, A. Gokaslan, J. Zhang, K. Chandu, T. Nguyen, I. Vasiljevic, S. Kakade, S. Song, S. Sanghavi, F. Faghri, S. Oh, L. Zettlemoyer, K. Lo, A. El-Nouby, H. Pouransari, A. Toshev, S. Wang, D. Groeneveld, L. Soldaini, P. W. Koh, J. Jitsev, T. Kollar, A. G. Dimakis, Y. Carmon, A. Dave, L. Schmidt, and V. Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2024a. URL https://arxiv.org/abs/2406.11794.

N. Li, A. Pan, A. Gopal, S. Yue, D. Berrios, A. Gatti, J. D. Li, A.-K. Dombrowski, S. Goel, G. Mukobi, N. Helm-Burger, R. Lababidi, L. Justen, A. B. Liu, M. Chen, I. Barrass, O. Zhang, X. Zhu, R. Tamirisa, B. Bharathi, A. Herbert-Voss, C. B. Breuer, A. Zou, M. Mazeika, Z. Wang, P. Oswal, W. Lin, A. A. Hunt, J. Tienken-Harder, K. Y. Shih, K. Talley, J. Guan, I. Steneker, D. Campbell, B. Jokubaitis, S. Basart, S. Fitz, P. Kumaraguru, K. K. Karmakar, U. Tupakula, V. Varadharajan, Y. Shoshitaishvili, J. Ba, K. M. Esvelt, A. Wang, and D. Hendrycks. The WMDP benchmark: Measuring and reducing malicious use with unlearning. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28525–28550. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/li24bc.html.

R. Li, J. Fu, B.-W. Zhang, T. Huang, Z. Sun, C. Lyu, G. Liu, Z. Jin, and G. Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023a.

X. Li, T. Zhang, Y. Dubois, R. Taori, I. Gulrajani, C. Guestrin, P. Liang, and T. B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 5 2023b.

Y. Li, Y. Ma, S. Yan, C. Zhang, J. Liu, J. Lu, Z. Xu, M. Chen, M. Wang, S. Zhan, J. Ma, X. Lai, D. Liu, Y. Luo, X. Bin, H. Ren, M. Han, W. Hao, B. Yi, L. Liu, B. Ma, X. Jia, X. Zhou, S. Qiao, L. Xiang, and Y. Wu. Model merging in pre-training of large language models. *ArXiv*, abs/2505.12082, 2025. URL `https://api.semanticscholar.org/CorpusID:278739754`.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

B. Y. Lin, R. L. Bras, K. Richardson, A. Sabharwal, R. Poovendran, P. Clark, and Y. Choi. Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2502.01100*, 2025.

B. Liu, S. Bubeck, R. Eldan, J. Kulkarni, Y. Li, A. Nguyen, R. Ward, and Y. Zhang. Tinygsm: achieving >80 URL `https://arxiv.org/abs/2312.09241`.

J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL `https://openreview.net/forum?id=1qvx610Cu7`.

M. Liu, S. Diao, X. Lu, J. Hu, X. Dong, Y. Choi, J. Kautz, and Y. Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint*, 2025a. URL `https://arxiv.org/abs/2505.24864`.

Q. Liu, X. Zheng, N. Muennighoff, G. Zeng, L. Dou, T. Pang, J. Jiang, and M. Lin. Regmix: Data mixture as regression for language model pre-training. *arXiv preprint arXiv:2407.01492*, 2024a.

W. Liu, X. Huang, X. Zeng, X. Hao, S. Yu, D. Li, S. Wang, W. Gan, Z. Liu, Y. Yu, Z. Wang, Y. Wang, W. Ning, Y. Hou, B. Wang, C. Wu, X. Wang, Y. Liu, Y. Wang, D. Tang, D. Tu, L. Shang, X. Jiang, R. Tang, D. Lian, Q. Liu, and E. Chen. Toolace: Winning the points of llm function calling. *ArXiv*, abs/2409.00920, 2024b. URL `https://api.semanticscholar.org/CorpusID:272368347`.

Z. Liu, A. Qiao, W. Neiswanger, H. Wang, B. Tan, T. Tao, J. Li, Y. Wang, S. Sun, O. Pangarkar, et al. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*, 2023c.

Z. Liu, T. Hoang, J. Zhang, M. Zhu, T. Lan, S. Kokane, J. Tan, W. Yao, Z. Liu, Y. Feng, R. Murthy, L. Yang, S. Savarese, J. C. Niebles, H. Wang, S. Heinecke, and C. Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *ArXiv*, abs/2406.18518, 2024c. URL `https://api.semanticscholar.org/CorpusID:270738094`.

Z. Liu, C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin. Understanding r1-zero-like training: A critical perspective. In *Conference on Language Modeling (COLM)*, 2025b.

Z. Liu, B. Tan, H. Wang, W. Neiswanger, T. Tao, H. Li, F. Koto, Y. Wang, S. Sun, O. Pangarkar, et al. Llm360 k2: Building a 65b 360-open-source large language model from scratch. *arXiv preprint arXiv:2501.07124*, 2025c.

A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

M. Luo, S. Tan, J. Wong, X. Shi, W. Y. Tang, M. Roongta, C. Cai, J. Luo, L. E. Li, R. A. Popa, and I. Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. `https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2`, 2025a. Notion Blog.

M. Luo, S. Tan, J. Wong, X. Shi, W. Y. Tang, M. Roongta, C. Cai, J. Luo, T. Zhang, L. E. Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 2025b.

Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

I. Magar and R. Schwartz. Data contamination: From memorization to exploitation. *ArXiv*, abs/2203.08242, 2022. URL `https://api.semanticscholar.org/CorpusID:247475929`.

I. Magnusson, N. Tai, B. Bogin, D. Heineman, J. D. Hwang, L. Soldaini, A. Bhagia, J. Liu, D. Groeneveld, O. Tafjord, N. A. Smith, P. W. Koh, and J. Dodge. Datadecide: How to predict best pretraining data with small experiments, 2025. URL `https://arxiv.org/abs/2504.11393`.

A. Mallen, A. Asai, V. Zhong, R. Das, H. Hajishirzi, and D. Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint*, 2022.

S. V. Marjanović, A. Patel, V. Adlakha, M. Aghajohari, P. BehnamGhader, M. Bhatia, A. Khandelwal, A. Kraft, B. Krojer, X. H. Lù, N. Meade, D. Shin, A. Kazemnejad, G. Kamath, M. Mosbach, K. Stańczak, and S. Reddy. Deepseek-r1 thoughtology: Let's think about llm reasoning, 2025. URL https://arxiv.org/abs/2504.07128.

H. Markovits, V. A. Thompson, and J. Brisson. Metacognition and abstract reasoning. *Memory & Cognition*, 43(4):681–693, 2015. ISSN 1532-5946. doi: 10.3758/s13421-014-0488-9. URL https://doi.org/10.3758/s13421-014-0488-9.

A. Matton, T. Sherborne, D. Aumiller, E. Tommasone, M. Alizadeh, J. He, R. Ma, M. Voisin, E. Gilsenan-McMahon, and M. Gallé. On leakage of code generation evaluation datasets. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13215–13223, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.772. URL https://aclanthology.org/2024.findings-emnlp.772/.

M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

Y. Meyer and D. Corneil. Nemotron-Personas-USA: Synthetic personas aligned to real-world distributions, June 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Personas-USA.

S. Mindermann, J. M. Brauner, M. T. Razzak, M. Sharma, A. Kirsch, W. Xu, B. Höltgen, A. N. Gomez, A. Morisot, S. Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pages 15630–15649. PMLR, 2022.

M. Miroyan, T.-H. Wu, L. King, T. Li, J. Pan, X. Hu, W.-L. Chiang, A. N. Angelopoulos, T. Darrell, N. Norouzi, and J. Gonzalez. Search arena: Analyzing search-augmented llms. *ArXiv*, abs/2506.05334, 2025. URL https://api.semanticscholar.org/CorpusID:279243096.

I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

J. Morrison, N. A. Smith, H. Hajishirzi, P. W. Koh, J. Dodge, and P. Dasigi. Merge to learn: Efficiently adding skills to language models with model merging, 2024. URL https://arxiv.org/abs/2410.12937.

MosaicML. Llm foundry - jeopardy dataset. https://github.com/mosaicml/llm-foundry/blob/main/scripts/eval/local_data/world_knowledge/jeopardy_all.jsonl, 2024. Accessed: 2024-11-10.

I. Moshkov, D. Hanley, I. Sorokin, S. Toshniwal, C. Henkel, B. Schifferer, W. Du, and I. Gitman. AIMO-2 Winning Solution: Building State-of-the-Art Mathematical Reasoning Models with OpenMathReasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

N. Muennighoff, A. M. Rush, B. Barak, T. L. Scao, A. Piktus, N. Tazi, S. Pyysalo, T. Wolf, and C. Raffel. Scaling data-constrained language models, 2025a. URL https://arxiv.org/abs/2305.16264.

N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling, 2025b. URL https://arxiv.org/abs/2501.19393.

D. Nathawani, I. Gitman, S. Majumdar, E. Bakhturina, A. Sunil Mahabaleshwarkar, , J. Zhang, and J. Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1.

E. Nelson, G. Kollias, P. Das, S. Chaudhury, and S. Dan. Needle in the haystack for memory based large language models, 2024. URL https://arxiv.org/abs/2407.01437.

M. Noukhovitch, S. Huang, S. Xhonneux, A. Hosseini, R. Agarwal, and A. Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models, 2024. URL https://arxiv.org/abs/2410.18252.

NVIDIA, , A. Basant, A. Khairnar, A. Paithankar, A. Khattar, A. Renduchintala, A. Malte, A. Bercovich, A. Hazare, A. Rico, A. Ficek, A. Kondratenko, A. Shaposhnikov, A. Bukharin, A. Taghibakhshi, A. Barton, A. S. Mahabaleshwarkar, A. Shen, A. Tao, A. Guan, A. Shors, A. Mandarwal, A. Mehta, A. Venkatesan, A. Sharabiani, A. Aithal, A. Poojary, A. Dattagupta, B. Buddharaju, B. Zhu, B. Simkin, B. Kartal, B. D. Rouhani, B. Chen, B. Ginsburg, B. Norick, B. Yu, B. Catanzaro, C. Wang, C. Truong, C. Mungekar, C. Patel, C. Alexiuk, C. Munley, C. Parisien, D. Su, D. Afrimi, D. Korzekwa, D. Rohrer, D. Gitman, D. Mosallanezhad, D. Narayanan, D. Rekesh, D. Yared, D. Pykhtar, D. Ahn, D. Riach, E. Long, E. Ning, E. Chung, E. Galinkin, E. Bakhturina, G. Prasad, G. Shen,

H. Qian, H. Elisha, H. Sharma, H. Ross, H. Ngo, H. Sahota, H. Wang, H. C. Shin, H. Huang, I. Cunningham, I. Gitman, I. Moshkov, J. Jung, J. Kautz, J. P. Scowcroft, J. Casper, J. Zhang, J. Zeng, J. Zhang, J. Xue, J. Huang, J. Conway, J. Kamalu, J. Cohen, J. Jennings, J. V. Vialard, J. Yi, J. Parmar, K. Briski, K. Cheung, K. Luna, K. Wyss, K. Santhanam, K. Kong, K. Pawelec, K. Anik, K. Li, K. Ahmadian, L. McAfee, L. Sleiman, L. Derczynski, L. Vega, M. R. de Melo, M. N. Sreedhar, M. Chochowski, M. Cai, M. Kliegl, M. Stepniewska-Dziubinska, M. Novikov, M. Samadi, M. Price, M. Boubdir, M. Boone, M. Evans, M. Bien, M. Zawalski, M. Martinez, M. Chrzanowski, M. Shoeybi, M. Patwary, N. Dhameja, N. Assaf, N. Habibi, N. Bhatia, N. Pope, N. Tajbakhsh, N. K. Juluru, O. Rybakov, O. Hrinchuk, O. Kuchaiev, O. Olabiyi, P. Ribalta, P. Subramanian, P. Chadha, P. Molchanov, P. Dykas, P. Jin, P. Bialecki, P. Januszewski, P. Thalasta, P. Gaikwad, P. Varshney, P. Gundecha, P. Tredak, R. K. Mahabadi, R. Patel, R. El-Yaniv, R. Rajan, R. Cheruvu, R. Shahbazyan, R. Borkar, R. Gala, R. Waleffe, R. Zhang, R. J. Hewett, R. Prenger, S. Jain, S. Kriman, S. Satheesh, S. Kaji, S. Yurick, S. Muralidharan, S. Narenthiran, S. Bak, S. Sameni, S. Han, S. Ramasamy, S. Ghosh, S. T. Sreenivas, S. Thomas, S. Diao, S. Gopal, S. Prabhumoye, S. Toshniwal, S. Ding, S. Singh, S. Jain, S. Majumdar, S. Singhal, S. Alborghetti, S. N. Akter, T. Kong, T. Moon, T. Hliwiak, T. Asida, T. Wang, T. Konuk, T. Vashishth, T. Poon, U. Karpas, V. Noroozi, V. Srinivasan, V. Korthikanti, V. Fugro, V. Kalluru, V. Kurin, V. Lavrukhin, W. U. Ahmad, W. Du, W. Byeon, X. Lu, X. Dong, Y. Karnati, Y. Choi, Y. Zhang, Y. Lin, Y. Fu, Y. Suhara, Z. Dong, Z. Li, Z. Zhu, and Z. Chen. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

NVIDIA AI. Nemotron-post-training-dataset-v1. https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1, 2025. Dataset.

J. Olieslagers, Z. Bnaya, Y. Li, and W. Ma. Backward reasoning through and/or trees to solve problems. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 46. Cognitive Science Society, 2024. URL https://escholarship.org/uc/item/9h4863xm. Retrieved from https://escholarship.org/uc/item/9h4863xm.

T. OLMo, P. Walsh, L. Soldaini, D. Groeneveld, K. Lo, S. Arora, A. Bhagia, Y. Gu, S. Huang, M. Jordan, N. Lambert, D. Schwenk, O. Tafjord, T. Anderson, D. Atkinson, F. Brahman, C. Clark, P. Dasigi, N. Dziri, M. Guerquin, H. Ivison, P. W. Koh, J. Liu, S. Malik, W. Merrill, L. J. V. Miranda, J. Morrison, T. Murray, C. Nam, V. Pyatkin, A. Rangapur, M. Schmitz, S. Skjonsberg, D. Wadden, C. Wilhelm, M. Wilson, L. Zettlemoyer, A. Farhadi, N. A. Smith, and H. Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

OpenAI. GPT-3.5 turbo, 2023a. URL https://platform.openai.com/docs/models/gp#gpt-3-5-turbo.

OpenAI. GPT-4 technical report. *ArXiv*, abs/2303.08774, 2023b. URL https://api.semanticscholar.org/CorpusID:257532815.

OpenAI. Gpt-5 system card. Technical report, OpenAI, Aug. 2025. Accessed: 2025-10-07.

A. Pal, L. K. Umapathi, and M. Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In G. Flores, G. H. Chen, T. Pollard, J. C. Ho, and T. Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR, 07–08 Apr 2022. URL https://proceedings.mlr.press/v174/pal22a.html.

R. Pandey. gzip predicts data-dependent scaling laws, 2024. URL https://arxiv.org/abs/2405.16684.

D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

A. Parrish, A. Chen, N. Nangia, V. Padmakumar, J. Phang, J. Thompson, P. M. Htut, and S. Bowman. BBQ: A hand-built bias benchmark for question answering. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2086–2105, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.165. URL https://aclanthology.org/2022.findings-acl.165/.

K. Paster, M. D. Santos, Z. Azerbayev, and J. Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023.

S. G. Patil, H. Mao, C. Cheng-Jie Ji, F. Yan, V. Suresh, I. Stoica, and J. E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only, 2023. URL https://arxiv.org/abs/2306.01116.

G. Penedo, H. Kydlíček, A. Lozhkov, M. Mitchell, C. Raffel, L. Von Werra, T. Wolf, et al. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *The Thirty-eight Conference on Neural Information Processing Systems; Datasets and Benchmarks Track*, 2024.

B. Peng, J. Quesnelle, H. Fan, and E. Shippole. Yarn: Efficient context window extension of large language models, 2023. URL https://arxiv.org/abs/2309.00071.

C. M. Pham, Y. Chang, and M. Iyyer. Clipper: Compression enables long-context synthetic data generation, 2025. URL https://arxiv.org/abs/2502.14854.

A. Piché, E. Kamaloo, R. Pardinas, and D. Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generatio. *arXiv preprint arXiv:2509.19128*, 2025.

J. Poznanski, A. Rangapur, J. Borchardt, J. Dunkelberger, R. Huff, D. Lin, C. Wilhelm, K. Lo, and L. Soldaini. olmOCR: Unlocking trillions of tokens in pdfs with vision language models. *arXiv preprint arXiv:2502.18443*, 2025a.

J. Poznanski, L. Soldaini, and K. Lo. olmOCR 2: Unit Test Rewards for Document OCR, 2025b. URL https://arxiv.org/abs/2510.19817.

PrimeIntellect. Synthetic-2. https://huggingface.co/datasets/PrimeIntellect/SYNTHETIC-2, 2025. Dataset.

V. Pyatkin, S. Malik, V. Graf, H. Ivison, S. Huang, P. Dasigi, N. Lambert, and H. Hajishirzi. Generalizing verifiable instruction following. *arXiv preprint arXiv:2507.02833*, 2025.

Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwq-32b: Embracing the power of reinforcement learning. https://qwenlm.github.io/blog/qwq-32b/, Mar. 2025. Model release blog.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506, 2020.

S. Reddy, D. Chen, and C. D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

P. Röttger, H. R. Kirk, B. Vidgen, G. Attanasio, F. Bianchi, and D. Hovy. Xstest: A test suite for identifying exaggerated safety behaviours in large language models. *arXiv preprint arXiv:2308.01263*, 2023.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr. 2020. doi: 10.1609/aaai.v34i05.6399. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

M. Sap, H. Rashkin, D. Chen, R. Le Bras, and Y. Choi. Social IQa: Commonsense reasoning about social interactions. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in*

Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4463–4473, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454.

D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

R. Schaeffer, B. Miranda, and S. Koyejo. Are emergent abilities of large language models a mirage? *Advances in neural information processing systems*, 36:55565–55581, 2023.

R. Shao, S. S. Li, R. Xin, S. Geng, Y. Wang, S. Oh, S. S. Du, N. Lambert, S. Min, R. Krishna, Y. Tsvetkov, H. Hajishirzi, P. W. Koh, and L. Zettlemoyer. Spurious rewards: Rethinking training signals in rlvr, 2025. URL https://arxiv.org/abs/2506.10947.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1671–1685, 2024.

D. Shi, J. Cao, Q. Chen, W. Sun, W. Li, H. Lu, F. Dong, T. Qin, K. Zhu, M. Liu, J. Yang, G. Zhang, J. Liu, C. Zhang, J. Wang, Y. E. Jiang, and W. Zhou. Taskcraft: Automated generation of agentic tasks. *ArXiv*, abs/2506.10055, 2025. URL https://api.semanticscholar.org/CorpusID:279318561.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

S. Singh, F. Vargus, D. Dsouza, B. F. Karlsson, A. Mahendiran, W.-Y. Ko, H. Shandilya, J. Patel, D. Mataciunas, L. O'Mahony, M. Zhang, R. Hettiarachchi, J. Wilson, M. Machado, L. S. Moura, D. Krzemiński, H. Fadaei, I. Ergün, I. Okoh, A. Alaagib, O. Mudannayake, Z. Alyafeai, V. M. Chien, S. Ruder, S. Guthikonda, E. A. Alghamdi, S. Gehrmann, N. Muennighoff, M. Bartolo, J. Kreutzer, A. Üstün, M. Fadaee, and S. Hooker. Aya dataset: An open-access collection for multilingual instruction tuning. *arXiv preprint arXiv:2402.06619*, 2024. URL https://arxiv.org/abs/2402.06619.

M. D. Skarlinski, S. Cox, J. M. Laurent, J. D. Braza, M. Hinks, M. J. Hammerling, M. Ponnapati, S. G. Rodriques, and A. D. White. Language agents achieve superhuman synthesis of scientific knowledge. *arXiv preprint arXiv:2409.13740*, 2024.

L. Soldaini and K. Lo. peS2o (Pretraining Efficiently on S2ORC) Dataset, 2023. URL https://github.com/allenai/pes2o.

L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, V. Hofmann, A. H. Jha, S. Kumar, L. Lucy, X. Lyu, N. Lambert, I. Magnusson, J. Morrison, N. Muennighoff, A. Naik, C. Nam, M. E. Peters, A. Ravichander, K. Richardson, Z. Shen, E. Strubell, N. Subramani, O. Tafjord, P. Walsh, L. Zettlemoyer, N. A. Smith, H. Hajishirzi, I. Beltagy, D. Groeneveld, J. Dodge, and K. Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research, 2024.

A. Souly, Q. Lu, D. Bowen, T. Trinh, E. Hsieh, S. Pandey, P. Abbeel, J. Svegliato, S. Emmons, O. Watkins, and S. Toyer. A strongreject for empty jailbreaks. 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 125416–125440. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e2e06adf560b0706d3b1ddfca9f29756-Paper-Datasets_and_Benchmarks_Track.pdf.

N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

D. Su, K. Kong, Y. Lin, J. Jennings, B. Norick, M. Kliegl, M. Patwary, M. Shoeybi, and B. Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2025a. URL https://arxiv.org/abs/2412.02595.

J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Y. Su, D. Yu, L. Song, J. Li, H. Mi, Z. Tu, M. Zhang, and D. Yu. Expanding rl with verifiable rewards across diverse domains. *arXiv preprint arXiv:2503.23829*, 2025b.

Z. Su, L. Pan, X. Bai, D. Liu, G. Dong, J. Huang, W. Hu, F. Zhang, K. Gai, and G. Zhou. Klear-reasoner: Advancing reasoning capability via gradient-preserving clipping policy optimization. *arXiv preprint arXiv:2508.07629*, 2025c.

Y. Sun, S. Hu, G. Zhou, K. Zheng, H. Hajishirzi, N. Dziri, and D. X. Song. Omega: Can llms reason outside the box in math? evaluating exploratory, compositional, and transformative generalization. *ArXiv*, abs/2506.18880, 2025. URL https://api.semanticscholar.org/CorpusID:280000246.

M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

A. Talmor, J. Herzig, N. Lourie, and J. Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023. URL https://huggingface.co/datasets/teknium/OpenHermes-2.5.

The Algorithms. The algorithms – python. https://github.com/TheAlgorithms/Python, 2025. GitHub repository, MIT License.

Together AI. RedPajama: An open source recipe to reproduce LLaMA training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

S. Toshniwal, W. Du, I. Moshkov, B. Kisacanin, A. Ayrapetyan, and I. Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

J. Toy, J. MacAdam, and P. Tabor. Metacognition is all you need? using introspection in generative agents to improve goal-directed behavior, 2024. URL https://arxiv.org/abs/2401.10910.

H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

J. Vendrow, E. Vendrow, S. Beery, and A. Madry. Do large language model benchmarks test reliability? *arXiv preprint arXiv:2502.03461*, 2025.

D. Wadden, K. Shi, J. Morrison, A. Naik, S. Singh, N. Barzilay, K. Lo, T. Hope, L. Soldaini, S. Z. Shen, et al. Sciriff: A resource to enhance language model instruction-following over scientific literature. *arXiv preprint arXiv:2406.07835*, 2024.

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024a.

Z. Wang, Y. Dong, O. Delalleau, J. Zeng, G. Shen, D. Egert, J. J. Zhang, M. N. Sreedhar, and O. Kuchaiev. Helpsteer2: Open-source dataset for training top-performing reward models. *arXiv preprint arXiv:2406.08673*, 2024b.

Z. Wang, F. Zhou, X. Li, and P. Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025.

J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2021.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

J. Wei, N. Karina, H. W. Chung, Y. J. Jiao, S. Papay, A. Glaese, J. Schulman, and W. Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024.

J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. In L. Derczynski, W. Xu, A. Ritter, and T. Baldwin, editors, *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL `https://aclanthology.org/W17-4413/`.

A. Wettig, K. Lo, S. Min, H. Hajishirzi, D. Chen, and L. Soldaini. Organize the web: Constructing domains enhances pre-training data curation, 2025a. URL `https://arxiv.org/abs/2502.10341`.

A. Wettig, K. Lo, S. Min, H. Hajishirzi, D. Chen, and L. Soldaini. Organize the web: Constructing domains enhances pre-training data curation. *arXiv preprint arXiv:2502.10341*, 2025b.

M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

J. Wu, W. Yin, Y. Jiang, Z. Wang, Z. Xi, R. Fang, L. Zhang, Y. He, D. Zhou, P. Xie, and F. Huang. WebWalker: Benchmarking LLMs in web traversal. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10290–10305, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.508. URL `https://aclanthology.org/2025.acl-long.508/`.

L. Wu, D. Zhu, G. Zhao, Z. Yu, J. Ran, X. Wong, L. Sun, and S. Li. LongAttn: Selecting long-context training data via token-level attention, 2025b. URL `https://arxiv.org/abs/2502.16860`.

M. Wu, Z. Zhang, Q. Dong, Z. Xi, J. Zhao, S. Jin, X. Fan, Y. Zhou, H. Lv, M. Zhang, et al. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. *arXiv preprint arXiv:2507.10532*, 2025c.

L.-C. Xiaomi, :, B. Xia, B. Shen, Cici, D. Zhu, D. Zhang, G. Wang, H. Zhang, H. Liu, J. Xiao, J. Dong, L. Zhao, P. Li, P. Wang, S. Yu, S. Chen, W. Wang, W. Ma, X. Deng, Y. Huang, Y. Song, Z. Jiang, B. Ye, C. Cai, C. He, D. Zhang, D. Zhang, G. Wang, H. Tian, H. Zhao, H. Qu, H. Xu, J. Shi, K. Bao, K. Fang, K. Zhou, K. Zhou, L. Li, M. Zhu, N. Chen, Q. Wang, S. Liu, S. Li, S. Gu, S. Ren, S. Liu, S. Deng, W. Zhuang, W. Lv, W. Yang, X. Zhang, X. Yong, X. Zhang, X. Song, X. Xu, X. Wang, Y. Yan, Y. Tu, Y. Tian, Y. Wang, Y. Yu, Z. Lin, Z. Song, and Z. Yue. MiMo: Unlocking the reasoning potential of language model – from pretraining to posttraining, 2025. URL `https://arxiv.org/abs/2505.07608`.

W. Xiong, J. Liu, I. Molybog, H. Zhang, P. Bhargava, R. Hou, L. Martin, R. Rungta, K. A. Sankararaman, B. Oguz, M. Khabsa, H. Fang, Y. Mehdad, S. Narang, K. Malik, A. Fan, S. Bhosale, S. Edunov, M. Lewis, S. Wang, and H. Ma. Effective long-context scaling of foundation models, 2023. URL `https://arxiv.org/abs/2309.16039`.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

A. Yang, B. Yu, C. Li, D. Liu, F. Huang, H. Huang, J. Jiang, J. Tu, J. Zhang, J. Zhou, J. Lin, K. Dang, K. Yang, L. Yu, M. Li, M. Sun, Q. Zhu, R. Men, T. He, W. Xu, W. Yin, W. Yu, X. Qiu, X. Ren, X. Yang, Y. Li, Z. Xu, and Z. Zhang. Qwen2.5-1M Technical Report, 2025b. URL `https://arxiv.org/abs/2501.15383`.

Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018. URL `https://api.semanticscholar.org/CorpusID:52822214`.

F. Yao, L. Liu, D. Zhang, C. Dong, J. Shang, and J. Gao. Your efficient rl framework secretly brings you off-policy rl training, Aug. 2025. URL `https://fengyao.notion.site/off-policy-rl`.

J. Ye, P. Liu, T. Sun, J. Zhan, Y. Zhou, and X. Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance, 2025. URL `https://arxiv.org/abs/2403.16952`.

H. Yen, T. Gao, M. Hou, K. Ding, D. Fleischer, P. Izsak, M. Wasserblat, and D. Chen. HELMET: How to evaluate long-context models effectively and thoroughly. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=293V3bJbmE`.

A. Young, B. Chen, C. Li, C. Huang, G. Zhang, G. Zhang, H. Li, J. Zhu, J. Chen, J. Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Y. Yue, Z. Chen, R. Lu, A. Zhao, Z. Wang, Y. Yue, S. Song, and G. Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In A. Korhonen, D. Traum, and L. Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472.

H. Zeng, J. Yang, Y. Zhang, B. Yu, S. Wang, Z. Liu, M. Sun, and T. Liu. Acecoder: Acing coder rl via automated test-case synthesis. *arXiv preprint arXiv:2502.01718*, 2025a. URL https://arxiv.org/abs/2502.01718.

Z. Zeng, H. Ivison, Y. Wang, L. Yuan, S. S. Li, Z. Ye, S. Li, J. He, R. Zhou, T. Chen, C. Zhao, Y. Tsvetkov, S. S. Du, N. Jaques, H. Peng, P. W. Koh, and H. Hajishirzi. Rlve: Scaling up reinforcement learning for language models with adaptive verifiable environments. *arXiv preprint 2511.07317*, 2025b.

L. Zha, J. Zhou, L. Li, R. Wang, Q. Huang, S. Yang, J. Yuan, C. Su, X. Li, A. Su, T. Zhang, C. Zhou, K. Shou, M. Wang, W. Zhu, G. Lu, C. Ye, Y. Ye, W. Ye, Y. Zhang, X. Deng, J. Xu, H. Wang, G. Chen, and J. Zhao. Tablegpt: Towards unifying tables, natural language and commands into one gpt. *arXiv preprint arXiv:2307.08674*, 2023. URL https://arxiv.org/abs/2307.08674.

W. Zhao, X. Ren, J. Hessel, C. Cardie, Y. Choi, and Y. Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024a.

Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

Y. Zhao, Y. Qu, K. Staniszewski, S. Tworkowski, W. Liu, P. Miłoś, Y. Wu, and P. Minervini. Analysing the impact of sequence composition on language model pre-training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 7897–7912. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.acl-long.427. URL http://dx.doi.org/10.18653/v1/2024.acl-long.427.

W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*, 2023.

F. Zhou, Z. Wang, N. Ranjan, Z. Cheng, L. Tang, G. He, Z. Liu, and E. P. Xing. Megamath: Pushing the limits of open math corpora. *arXiv preprint arXiv:2504.02807*, 2025.

J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

A Appendix

Acknowledgements

A.1   Base Model Additional Data Details: Pretraining

Layers	32 / 64	**Gradient Clipping**	1.0
Hidden Size (d_{model})	4096 / 5120	**Z-Loss Weight**	10^{-5}
Q Heads	32 / 40	**Weight Decay on Embeddings**	No
KV Heads	32 / 8	**Sliding Window Attention**	3/4 of layers
Activation	SwiGLU	**RoPE Scaling**	YaRN
QKV Normalization	QK-Norm	**RoPE** θ	$5 \cdot 10^5$
Layer Norm	RMSNorm	**Layer Norm Applied to**	Outputs

Table 30 Architecture for OLMO 3 7B / 32B.

	Pretraining	Mid-training	Long-context Extension
Learning Rate Schedule	Modified Cosine (see Figure 3)	Linear decay	Linear decay
LR Warmup from 0	2000 steps	0 steps	200 steps
Peak LR	3.0×10^{-4}	2.074×10^{-4}	2.074×10^{-4}
Final LR	3.0×10^{-5}	0	0
Batch Size in Instances	512	256	64
Sequence Length	8,192	8,192	65,536
Batch Size in Tokens	4,194,304	2,097,152	4,194,304
Total Training Tokens	5.9T	100B	50B
Peak Training Temperature ($\frac{LR^2}{bsz}$)	2.146×10^{-14}	2.051×10^{-14}	1.026×10^{-14}

Table 31 Training hyperparameters for each stage of OLMO 3 7B.

	Pretraining	Mid-training	Long-context Extension
Learning Rate Schedule	5.9T cosine truncated at 5.7T tokens	100B cosine	Linear decay
LR Warmup from 0	2000 steps	0 steps	200 steps
Peak LR	6.0×10^{-4}	2.071×10^{-4}	2.071×10^{-4}
Final LR	6.0×10^{-5}	0	0
Batch Size in Instances	1,024	512	128
Sequence Length	8,192	8,192	65,536
Batch Size in Tokens	8,388,608	4,194,304	8,388,608
Total Training Tokens	5.7T	100B	100B
Peak Training Temperature ($\frac{LR^2}{bsz}$)	4.292×10^{-14}	1.023×10^{-14}	5.113×10^{-15}

Table 32 Training hyperparameters for each stage of OLMO 3 32B. Compared to the 7B, the batch size is doubled in all steps, as is the number of tokens in the long-context extension stage.

Figure 28 Cross-entropy loss and total gradient norm during pretraining for OLMO 3. For readability, gradient norm plots were produced using an exponential moving average with a window size of 20 steps.

A.1.1 CommonCrawl

The majority of our pretraining corpus comes from CommonCrawl (Common Crawl Foundation). We start with 104 dumps, starting with `CC-MAIN-2013-20` and ending with `CC-MAIN-2024-51`, roughly covering dates from mid-2013 until late 2024. We linearize the WET files provided by Commoncrawl using Resiliparse, yielding an initial pool composed of 252.6B documents.

Next we apply a pipeline of heuristic filtering steps to further prune down the dataset to a size amenable for pretraining. Our steps essentially follows that of DCLM (Li et al., 2024a), with a few small differences. We start with URL-based filtering, identifying and removing documents that have URLs that have banned words or subwords from the blacklists used by FineWeb (Penedo et al., 2024) and RefinedWeb (Penedo et al., 2023). This step removes roughly 1% of the data pool. Then we apply the DCLM collection of heuristic filters, roughly targeting and removing: i) very short documents, ii) very long documents, iii) documents with not enough alphanumeric characters, and iv) documents with large amounts of internal repetition. Then we modify and remove any lines/paragraphs in each document that have i) too many numeric characters, or ii) any boilerplate phrases such as "items in cart" or "read more..."; and then we fully remove any documents that have been obliterated by these line-specific removals. Then we apply a FastText English language filter, mirroring DCLM and using a threshold of 0.65 to identify documents as containing English text. Finally, we apply a subset of the rules for identifying questionable sentences from MADLAD-400 (Kudugunta et al., 2023). Ablation tests show that only rules 2 and 5 from MADLAD improve dataset quality, targeting sentences that have a large number of capitalized words or contain a "cursed regex". If the number of sentences in the document is less than 5 or if at least 20% of sentences are questionable, we remove the document from the corpus.

Overall, the heuristic steps remove 76% of the total pool, and the English filtering step removes an additional 2.5% of the pool. This leaves a pool of 38.7B documents, attaining a survival rate of 15.1%. While each of these described steps are incorporated into the DCLM processing pipeline, we note that these heuristic filters are commutative and that the English filtering is the slowest step, so efficiency gains can be attained by putting the language filtering step at the end. We spent a total of 1030 i4i.32xlarge ec2 hours in this step, incurring a cost of approximately $11,300. An exact breakdown of how much time was spent in each step is provided in Table 33.

Pipeline Step	Docs Removed (B)	% of pool removed	% of total time
URL Filters	2.3	0.9	1.68
Length Filters	103.4	40.42	8.03
Symbol Filters	56.5	22.1	4.13
Internal Repetition	32.1	12.53	31.41
Line Modifiers	7.1	2.79	10.0
English Filter	6.2	2.44	14.3
MadLad Filters	9.3	3.65	5.87
Quality Classifiers	0.0	0.0	24.58

Table 33 Web data processing step cost and removal breakdown during the heuristic processing steps. We started with 252.6B documents and ended with 38.7B documents for a total removal rate of 84.9%. This procedure took, in aggregate, approximately 1,030 hours on i4i.32xlarge EC2 instances.

A.1.2 Deduplication

As described in the main paper, we apply a three-stage deduplication pipeline to our dataset, with each stage targeting progressively more nuanced forms of redundancy: (i) global exact deduplication based on document content hashes to remove identical copies, (ii) 32-way sharded MinHash deduplication with exact Jaccard similarity verification to remove near-duplicate documents, and (iii) 56-way sharded fuzzy suffix array deduplication to eliminate repeated boilerplate text. We note that while applying exact deduplication before MinHash deduplication is technically redundant, exact deduplication is substantially more efficient computationally; hence this two-pass approach is much faster overall. For the exact and MinHash deduplication stages, we utilize the Duplodocus tool[45], and for the suffix array deduplication stage, we employ `bsade`[46].

Exact Deduplication   We perform exact deduplication in two sequential passes. During the heuristic filtration pipeline, we annotate each document with a 128-bit hash computed from the document text. We then apply an initial deduplication step to each of the 104 processed CommonCrawl dumps individually, arbitrarily retaining one copy of each document per dump. This within-dump deduplication removes 24% of the surviving document pool.

Following this, we aggregate all documents globally and perform a second exact deduplication pass across the entire corpus, again arbitrarily keeping one copy of each document. This global pass removes an additional 43% of the surviving pool. In total, exact deduplication eliminates 66% of the input documents, reducing the corpus to 12.7 billion documents for subsequent MinHash processing.

MinHash Fuzzy Deduplication   We partition the 12.7 billion document corpus resulting from exact deduplication into 32 shards of approximately equal size and perform MinHash deduplication independently on each shard. Our MinHash procedure broadly follows the approach outlined in (Lee et al., 2022). We tokenize documents using the p50k tokenizer and construct sets of 5-gram token sequences. We then apply a MinHash locality-sensitive hashing scheme with 26 bands of size 11, configured to target a Jaccard similarity threshold of 0.80.

For any pair of documents that share at least one matching bucket, we treat them as connected by an edge in graph-theoretic terms. We construct a graph from the union of all such edges and identify connected components within this graph. Each document in a connected component is then annotated with a unique identifier for that component.

In a second verification phase, we explicitly compute pairwise Jaccard similarities within each MinHash-identified cluster to eliminate false positives. For this verification, we use 3-gram token sequences. Our approach varies based on cluster size: for connected components containing 500 or more documents, we apply a more stringent MinHash configuration using 200 bands of size 31; for components with fewer than 500

---

[45] https://github.com/allenai/duplodocus
[46] https://github.com/liujch1998/bsade

documents, we perform exhaustive pairwise Jaccard similarity checks and generate final duplicate clusters from these results.

After annotating all documents according to their true Jaccard similarity with other documents in the corpus, we retain only the most recent version of each document based on crawl date, removing all earlier duplicates. This complete MinHash deduplication procedure eliminates 24% of the input documents, leaving 9.8 billion documents in the pool.

Suffix Array Deduplication   In the final deduplication stage, we employ suffix arrays to identify and remove substrings that appear repeatedly throughout the dataset. We partition the 9.8 billion document corpus into 56 shards of roughly equal size and run suffix array deduplication independently on each shard.

For each shard, we construct a suffix array and identify every byte sequence of length 500 or greater that appears at least twice in the shard. We then apply a novel "fuzzy suffix array" removal strategy that considers contiguous text spans within each document. Specifically, if a text span is bounded on both sides by 500-byte sequences that appear multiple times in the suffix array, and at least 80% of the span is covered by such repeated sequences, we remove the entire span. This strategy targets cases where naive suffix array deduplication would leave short, unique fragments interspersed between removed substrings. For text that does not meet this bookended criterion, we remove all individual occurrences of repeated 500-byte sequences.

After these three rounds of deduplication—exact, MinHash, and suffix array—we arrive at a final corpus of 9.7 billion documents.

A.1.3   Topic Classification

Category	F1	Prec.	Rec.	Category	F1	Prec.	Rec.
Finance and Business	0.755	0.758	0.751	Travel and Tourism	0.781	0.780	0.782
Home and Hobbies	0.748	0.704	0.797	Crime and Law	0.735	0.747	0.724
Entertainment	0.801	0.773	0.832	Software	0.666	0.696	0.639
Sports and Fitness	0.870	0.850	0.890	Literature	0.759	0.801	0.721
Politics	0.788	0.786	0.790	Games	0.823	0.867	0.783
Health	0.822	0.824	0.820	Transportation	0.777	0.786	0.768
Education and Jobs	0.706	0.789	0.638	Religion	0.808	0.833	0.785
Science, Math and Technology	0.679	0.665	0.693	Electronics and Hardware	0.743	0.730	0.757
Social Life	0.628	0.609	0.649	Software Development	0.687	0.613	0.781
Fashion and Beauty	0.845	0.845	0.845	Industrial	0.710	0.691	0.731
Food and Dining	0.878	0.860	0.896	History and Geography	0.630	0.698	0.574
Art and Design	0.670	0.668	0.672	Adult Content	0.700	0.894	0.575
Overall (N=20,000): Precision = 0.762, Recall = 0.762							

Table 34 Performance of FastText WebOrganizer topic classifier on the held out sample of 20,000 documents used in the original WebOrganizer paper.

After strict rounds of deduplication, we partition our data according to topic using the 24 topic categories introduced in WebOrganizer (Wettig et al., 2025a). Rather than using the 140M parameter topic classifier used by WebOrganizer, we train a FastText classifier[47] to support cost-effective topic classification at scale. To train this classifier, we use the Llama-labeled training data used to train the original WebOrganizer category as well as an extra 506,746 examples with topics labeled by a combination of gpt-4.1 and o4-mini. The performance of this classifier is outlined in Table 34.

---

[47] https://huggingface.co/allenai/dolma3-fasttext-weborganizer-topic-classifier

A.1.4 CommonCrawl Mixing

We perform a hierarchical mixing procedure on our data. Our procedure OLMIX (Chen et al., 2025a) generates prescriptions for which percentage of the training mix should come from each topic or source, but offers no guidance on the quality composition within each topic. While prior works, such as DCLM (Li et al., 2024a) use a quality classifier to flatly filter data as high-quality or not, we take a more fine-grained approach and perform selective up and down-sampling within each WebOrganizer topic depending on the quality signal. This section formalizes the search procedure we use to generate these upsampling curves.

Problem Formulation: We discuss this procedure in more general terms: consider a category with X tokens, partitioned into Q strictly ordered quality buckets, where the qth bucket contains X_q tokens. Further assume that OLMIX prescribes that Z tokens be taken from this category, and that at no point do we want to upsample any quality bucket more than M times. This equates to a search problem, where we need to take Z_q tokens from the qth bucket such that $\sum_q Z_q = Z$ and $\forall q,\ Z_q/X_q \leq M$.

Parameterizing the Solution Space: To reduce the dimensionality of this search space, we make a modeling choice, where we search over a family of functions that control the upsampling ratio that meets the following criteria:

- Every function in the family is convex and monotonic.
- The functions are defined on the interval $[0,1]$, which can be normalized to the token counts later.
- We are able to control the integral such that $\int_0^1 f(x)dx = Z/X$.
- We can control the maximum average value of any one bucket. Suppose the qth bucket of data is arranged on the x-axis from $[a,b]$, then the maximum upsampling constraint correlates to the inequality $\frac{1}{b-a}\int_a^b f(x)dx \leq M$.
- We have the option to filter out the lowest quality buckets, i.e. $\int_0^a f(x)dx = 0$.

One such family of functions that meets these criteria is the family of truncated power-exponential functions:

$$f_{p,\lambda}(x) = \begin{cases} 0, & \text{for } x < a \\ C(x-a)^p \cdot e^{\lambda(x-a)}, & \text{for } x \geq a \end{cases}$$

Specifically, this becomes a feasibility problem for each topic of the data, where we search over parameters p,λ,C such that the constraints

- (Token yield is satisfied) $\int_0^1 f_{p,\lambda}(x)dx = Z/X$.
- (Maximum upsampling ratio is honored) $\frac{1}{b}\int_{1-b}^1 f_{p,\lambda}(x)dx \leq M$.
- (Function is monotonic) $\lambda \geq 0$.

are satisfied. The maximum upsampling constraint has been simplified such that, assuming monotonicity, the most upsampled quality bucket would be the highest-quality one, with an assumed data proportion of b.

Implementation Details: For each WebOrganizer topic, we set the maximum upsampling ration to be $M = 7$ and also throw away the bottom 40% in terms of quality, $a = 0.40$. Then we numerically solve for feasible p,λ,C. If the qth quality bucket spans from the q^- percentile to the q^+ percentile of the data, then the upsampling ratio for this bucket of data should be $\frac{1}{q^+-q^-}\int_{q^-}^{q^+} f(x)dx$.

A.1.5 Validating Quality Upsampling and Mixing

We validate our quality upsampling curves and mixing methodology both individually and jointly using small-scale 1B parameter models trained on 100B tokens. Our validation consists of three experiments:

	QA$_\text{Easy}$	Math$_\text{Easy}$	Code$_\text{Easy}$
Natural Distribution	100.70	71.87	59.20
QA-heavy Mix	**97.19**	64.34	53.45
Math-heavy Mix	97.98	**58.64**	49.65
Code-heavy Mix	98.64	61.85	**48.05**
OLMIX	99.54	61.69	48.94

Table 35 Token-constrained mixing allows optimizing different evaluation objectives. We use our swarms to optimize a QA-, Math- and Code-heavy data mix and train 1B models to 100B tokens. Results are on the OLMOBASEEVAL Easy suite (§3.3, lower is better).

	QA$_\text{Easy}$	Math$_\text{Easy}$	Code$_\text{Easy}$
Top 50% (1.1x repeat)	104.19	86.28	94.34
Top 30% (1.8x repeat)	103.10	86.97	88.00
Top 10% (5.6x repeat)	104.14	85.83	93.89
Top 5% (11.1x repeat)	106.53	84.31	93.02
OLMO 3 Upsampling	**99.95**	**74.04**	**71.88**

Table 36 Quality-aware upsampling outperforms naive data filtering. We simulate data-constrained training using 1B models trained to 100B tokens where we match the repetition of a 4.51T training run. Results are on the OLMOBASEEVAL Easy suite (§3.3, lower is better).

Targeted Mixing: We first verify that our mixing methodology can successfully optimize for specific prediction targets. Using our swarm optimization procedure, we create mixes optimized for three different objectives: the QA average, Math average, and Code average from OLMOBASEEVAL. We compare these targeted mixes against both the natural data distribution and the final OLMO 3 mix. Table 35 demonstrates that our swarm optimization successfully adapts the data distribution to match specific capability targets. While the final OLMOBASEEVAL mix exhibits slightly higher (worse) BPB scores than task-specific mixes due to necessary trade-offs across multiple objectives, it substantially outperforms the natural distribution.

Quality-Aware Upsampling   Next, we demonstrate that quality-aware upsampling outperforms naive quality-based filtering. To simulate a data-constrained 4.51T token training run, we compare different data selection strategies in Table 36. For the filtering baselines, we select the top percentiles from our vigintile quality buckets and match the resulting repetition factor that would occur when training on 100B tokens drawn from a theoretical 4.51T pool. For the upsampling approach, we apply the same methodology but set the target pool size to 100B tokens directly. Our results show that quality-aware upsampling consistently outperforms flat filtering across all repetition factors.

Reconciling Upsampling and Mixing.   Finally, we evaluate how to best combine our mixing and upsampling methodologies, which address complementary aspects of data selection. Data mixing determines the distribution across topics, while quality upsampling determines the distribution within a single source. To conceptualize this, imagine the dataset as a two-dimensional matrix of buckets where rows represent WebOrganizer topics and columns represent the quality buckets. Then the mixing strategy can be thought of as imposing row-wise (topic) constraints only. The quality-aware upsampling experiments in the preceding paragraph impose column-wise (quality) constraints only.

We considered several techniques that did not work quite as well as the truncated power-exponential forms described in § A.1.4. On one hand, the OLMIX framework samples data from each topic (row) according only to the natural quality distribution. On the other, quality upsampling samples data from each quality bucket (column) and does not consider reweighting topic distributions. For a theoretical target token yield, each of these strategies prescribes a target token count to be taken from each `(topic, quality)` bucket. Naive ways to rectify these strategies is to take an arithmetic or geometric mean between the target token counts from each bucket. We also note that the theoretical framework defining upsampling curves above is not necessarily restricted to the concept class of truncated power-exponential families. We could just as easily consider the family of exponential functions like $f_\lambda(x) = Ce^{\lambda(x-a)}$. Upon considering each of these techniques on small 1B models, we found that the truncated power-exponential family performed the best. Results are contained in Table 37.

A.2   Base Model Additional Data Details: Midtraining

This section provides further detail on curation processes for DOLMA 3 DOLMINO MIX. Additional replication resources, including prompts for synthetic data generation, are available in the Dolma3 GitHub repo.

	QA$_{Easy}$	Math$_{Easy}$	Code$_{Easy}$
Mixing Only	100.51	77.78	87.21
Quality Upsampling Only	102.21	82.08	80.88
Arithmetic Mean	100.38	79.15	82.83
Geometric Mean	100.42	78.18	81.34
Truncated exponential family	100.17	78.24	78.72
Truncated power-exponential family (OLMO 3)	**99.31**	**75.81**	**78.26**

Table 37 Different methods of combining quality-aware upsampling and token-constrained mixing to arrive at the final OLMO 3 pre-train mix. Results are on the OLMOBASEEVAL Easy suite (§3.3, lower is better).

Model	#Toks Seen (B)	#Toks Total (B)	MMLU	Math	MATH	GSM8K
tinyMATH (PoT)	0.24	0.24	-2.90	**16.58**	**20.70**	**25.33**
tinyMATH (MIND)	0.90	0.90	-1.75	11.62	12.48	14.80
tinyMATH (Both)	1.15	1.15	-1.68	9.98	11.40	12.07
CraneMath	4.34	4.34	0.01	4.86	4.26	6.32
SwallowMath	3.65	3.65	**0.33**	4.84	4.38	6.72
Dolminos Math	5.00	10.70	-0.60	4.68	2.08	7.65
MegaMatt	2.69	21.78	0.32	3.39	3.91	4.85
MM-Web-Pro	5.00	15.10	0.09	2.31	1.92	3.49
MM-Web-Pro-Max	5.00	73.85	-0.10	1.70	1.40	2.67
FineMath4+	6.89	9.61	0.03	1.51	1.21	2.19
MM-Web	5.00	263.90	0.03	1.30	0.69	2.16

Table 38 Results from math microanneals, with normalized per-token differences in scores relative to pre-anneal baseline. All anneals were run with a 50/50 mixture of web text data and the high quality data source. Numbers were arrived at by taking the difference from the pre-anneal baseline *and dividing by the number of tokens seen during training.*

A.2.1 Math Capabilities

Similar to Olmo 2 (OLMo et al., 2024), we take particular care to curate math-specific mixes of data during the midtraining phase of training. In this section we discuss some of the procedures used to generate, as well as validate, the math-specific data sources. It should be noted, that while there has been a flurry of research on high-quality, open-source, STEM-focused datasets, many of these are synthetic data generated using LLama-models, which carry with them a restrictive Llama license. We produce several reproductions of these with more permissive licensing and urge the community to take care in the licensing of the data they release if they wish to see adoption for research or commercial purposes.

TinyMATH In OLMo2, great strides were made in performance on the GSM-8K (Cobbe et al., 2021) dataset by generating synthetic math problems seeded from the original GSM training set, and then generating both python code (PoT) and natural language discussions of solutions (MIND). We adopt a similar strategy here, to target the MATH dataset (Hendrycks et al., 2021c). Namely, we adopt the TinyGSM protocol (Liu et al., 2023a) and prompts to generate 100 new problems for each existing MATH problem, and then generate pythonic solutions for each of these new problems. Then we apply the MIND rewrite prompt (Akter et al., 2024), using the two-student and problem-solving variants. This yields the PoT dataset (241M tokens) and the MIND dataset (899M tokens). To assess the potency of these new datasets, we ran annealing runs and evaluated fine-grained math related benchmarks as well as MMLU, to keep an eye on generalization. These results are summarized in TABLE:

CraneMath SwallowMath (Fujii et al., 2025) is a 2.3 Billion token dataset, generated from rewriting FineMath4+ (Allal et al., 2025). Unfortunately the data was rewritten using a Llama model, which would require that any model trained on this data would need to have "Llama" in the name, according to the Llama Community License. To provide truly open data, we mirror the generation of this dataset, but use Qwen3-32B Yang et al. (2025a) to rewrite FineMath4+ using the prompt presented in the SwallowMath paper. This yields a 5.62B token dataset we refer to as CraneMath. Compared to the 9.6B tokens contained in FineMATH4+, CraneMath is a distillation into fewer tokens, but not as few as SwallowMath (2.3B) – we posit that this is because using Qwen3 as a rewrite model is slightly "chattier" than Llama.

To evaluate performance of this rewrite procedure, we ran several anneals, starting from a base model that had seen 6T tokens of our pre-training mix, we ran several anneals, always with 50% token from the pretraining mix and 50% tokens from the data-source of interest. In the case where the anneals have different token counts, driving the learning rate linearly down to the same final learning rate. Then we compare the following runs: i) The pre-anneal baseline, ii) FineMath4+, but just an incomplete subset; iii) the original SwallowMath dataset; iv) our version, CraneMath; v) two copies of CraneMath; vi) a copy of CraneMath and all their original documents from FineMath4+.

MegaMatt OctoThinker (Wang et al., 2025) generated a 70B token data pool dubbed Megamath-Web-Pro-Max, intended to be a rewrite of LLM360's MegaMath data pool (Liu et al., 2023c), with quality mirroring that of the MegaMath-Web-Pro quality. Again, unfortunately, MegaMath-Web-Pro-Max was rewritten using Llama, and an independent recreation needed to be performed for fully open usage in training. Since our initial ablations showed that the Megamath-Web-Pro-Max pool wasn't as high of quality as, say, SwallowMath, we didn't need a recreation of the full 70B pool. Instead, we generated a recreation of just the documents from Megamath-Web-Pro-Max that occured in CommonCrawl dumps from dump `CC-MAIN-2023-23` and later, since more recent data was shown in the OctoThinker paper to be of higher quality. We ultimately generated 3.9B tokens of data, dubbed MegaMatt. To verify the efficacy, we ran ablations on: i) MegaMath-Web, ii) MegaMath-Web-Pro-Max (both to 10B and 25B tokens), and iii) MegaMatt.

OMR Rewrites Inspired by the success of Nvidia's OpenMathReasoning dataset on the AIO-2 Kaggle competition, we experimented with various rewrites sourced from AoPS forums Moshkov et al. (2025). See Dolma3 repo for further details.

Key Findings and Results We summarize the annealing results for the math datasets in Table 38. Each value reflects the change in the evaluation score relative to the pre-anneal baseline, normalized by the number of training tokens. Presenting the results this way highlights several distinct tiers of math-data quality, stratified by the effect-per-token. Notably, these quality tiers anticorrelate with the number of available tokens: the highest-quality sources are also the smallest. While it is true that there are diminishing returns of evaluation scores as more tokens are added, we claim that amongst these high-quality data sources, some higher quality than others.

At the top of the quality-spectrum are the tinyMATH variants. Although each contains less than a billion tokens, they deliver the strongest improvement per token – this is perhaps not surprising as these tokens were specifically crafted to augment the MATH evaluation score. Next in the tier-list of quality are the synthetic rewrites of natural high-quality data: the Crane, SwallowMath and MegaMatt sources which are each rewrites of FineMath4+ and MegaMathWeb-Pro. These provide a markedly weaker lift to the math evaluation metrics than the tinyMATH variants but also have a much larger pool of tokens to draw from. Finally, the largest data sources, including those of naturally occurring data such as FineMath4+ and MegamathWeb, also yield improvements, but their effect-per-token is noticeably smaller than that of the highly curated synthetic data. Finally we note that the effect of math midtraining on MMLU is generally neutral to negative, but is more strongly negative the more targeted the data pool is to Math evals, suggesting "overcooking", where increased specialization comes at the expense of broader general-purpose performance.

A.2.2 Code Capabilities

During pretraining, we relied entirely on stack-edu (Allal et al., 2025) for providing coding data. This data came in the form of naturally-occurring source code from github scraps with limited extra preprocessing.

During midtraining, we focused on improving Python and code-completion capabilities. To this end, we incorporated 10B tokens of FIM-transformed data form the same source as the pretraining code mixture. Inspired by improvements in math metrics by incorporating synthetic data, we also created a fully-open replica of SwallowCode (Fujii et al., 2025), which we call CraneCode.

CraneCode Of the off-the-shelf synthetic code data sources we considered, SwallowCode provided the greatest lift to coding evaluation metrics. Unfortunately, SwallowCode was generated using Llama models and thus had the less-permissive Llama license attached. We created a replica of SwallowCode by starting with just the python files from The Stack v2 Smol (Lozhkov et al., 2024), and applying the compilation and linting filters just as in SwallowCode. Then we applied a two-stage rewriting process, first to generate code data that is more compliant to the python style guides (SGCR), and then to generate optimized code (SCOR); both using the prompts from the original SwallowCode paper and Qwen/Qwen2.5-Coder-32B-Instruct (Qwen et al., 2024). To verify the the quality of the reproduced dataset, we ran several anneals, where results are displayed in Table 39.

Model	#Tokens	Crux-Eval	HumanEval	MBPP	MMLU
CraneCode (25B)	18.87B	35.92	**35.06**	31.72	54.30
CraneCode SGCR	18.87B	**41.75**	33.78	**36.76**	54.18
SwallowCode	10.0B	35.74	31.80	34.67	55.03
CraneCode (10B)	10.0B	33.28	26.51	34.94	54.98
Pre-anneal Baseline	N/A	35.46	21.51	27.11	**56.60**

Table 39 Microanneal results for CraneCode ablations. For each annealing run, we ran with a 50/50 mixture of web text and high-quality synthetic code data. We note several observations: 1) Both SwallowCode and CraneCode provide a lift to coding evaluation metrics at the expense of MMLU metrics; 2) SwallowCode provides a larger lift normalized for tokens than the CraneCode dataset; 3) CraneCode continues to provide lift to HumanEval as more data is provided, indicating that this data source is not yet exhausted.

A.2.3 Reasoning Capabilities

Meta-reasoning Recent work demonstrates that structured meta reasoning capabilities present during pre-training and mid-training serve as the foundation for successful reinforcement learning in complex reasoning tasks. Gandhi et al. (2025) showed that models exhibiting verification and backtracking behaviors during base training achieved dramatically superior performance trajectories during mathematical reasoning RL. Therefore, we begin by identifying structured reasoning capabilities that are critical for mathematical problem-solving. We select seven core capabilities that are foundational to mathematical and programming expertise: self-awareness (Toy et al., 2024; Callaway et al., 2022), self-evaluation (Fleming and Daw, 2017), goal management (Ackerman and Thompson, 2017; Griffiths et al., 2019), hierarchical organization (Haupt, 2018), backward chaining (Olieslagers et al., 2024), backtracking and conceptual reasoning (Markovits et al., 2015). We then design specific tasks that systematically target these capabilities, as shown in Table 40, and 41. For instance, Math Error Recovery specifically targets self-awareness, verification, and backtracking by requiring models to experience authentic mistakes and demonstrate recovery processes. Strategy Selection focuses purely on meta-cognitive choice processes, while Conversation Generation integrates all capabilities through educational dialogue. For data generation, we start with existing math (Luo et al., 2025a; Moshkov et al., 2025) and coding (Li et al., 2023a; Hendrycks et al., 2021a; Ahmad et al., 2025) problems and their corresponding correct answers. Following Pandalla dataset[48], we automatically augment each problem with detailed annotations[49] covering 'problem classification', 'difficulty analysis', 'solution approaches', 'common pitfalls', and 'verification methods'. These rich annotations serve as inputs for our capability-targeted tasks. For example, the 'common pitfalls' field directly informs math error recovery generation, while steps in 'solution approach' provides structure for backward chaining tasks. Using the annotated datasets as foundation, we employ `GPT-4.1` and `o4-mini` to generate training data at scale for each capability-targeted task.

---

[48] https://huggingface.co/datasets/pandalla/pandalla-math-dataset-v1.0

[49] We provide the problem and the correct answer as inputs to `o4-mini` with high reasoning, to synthesize the annotations following the Pandalla-math annotation schema.

Task	Meta Capabilities
Math error recovery	Self-awareness, verification, backtracking
Choosing the technique to use	Strategy selection
Difficulty estimation & self-awareness prompts	Self-evaluation
Steps generation	Goal management, hierarchical organization
From answer, generate steps backwards	Backward chaining
Conversation generation	All capabilities (tagging)
Reason about necessary concepts and how they connect	Conceptual reasoning

Table 40 Meta reasoning capabilities across mathematical tasks

Task	Meta Capabilities
Code error recovery (single-turn)	Self-awareness, verification, backtracking
Code error recovery (multi-turn)	Self-awareness, verification, backtracking
Planning the solution	Strategy selection, goal management
Solution implementation	Conceptual-level processing, hierarchical organization
Code quality evaluation (high/low)	Self-evaluation
Difficulty estimation	Self-evaluation, self-awareness
Unit test walkthrough	Goal management, verification

Table 41 Meta reasoning capabilities across coding tasks

Existing thinking traces   The full list of existing thinking traces is as follows:

1. **General Reasoning Mix** is a compilation of three existing datasets: GeneralThought-430K[50], OpenThoughts-114k (Guha et al., 2025b), and Open-R1-Math-220k[51]. The resulting dataset contains questions, thinking traces, and answers for topics spanning math, code, natural sciences, humanities, social sciences, and puzzles.

2. **Gemini Reasoning Traces**, introduced by Muennighoff et al. (2025b), contains thinking traces covering domains of math, astronomy, biology, chemistry, computer science, geography, physics, English, law, logic, and more.

3. **OpenThoughts2 Reasoning Traces** from Guha et al. (2025b) contains thinking traces in domains of math, science, code, and puzzles.

4. **Llama Nemotron Reasoning Traces** (Bercovich et al., 2025) contains thinking trace data for math, code, general reasoning, and instruction following.

5. **QwQ Reasoning Traces** consists of the QwQ subset of the OpenMathReasoning dataset (Moshkov et al., 2025).

Filtering steps included subselecting for permissively-licensed generations, filtering to remove empty and truncated responses, performing checks of verifiable claims and safety, filtering overt LLM self-references, filtering heavily repeated sentences, paragraphs, and phrases, and remove reasoning traces consisting of greater than 5% Chinese characters.

A.3   Base Model Additional Evaluation Details

The OLMoBaseEval suite expands on the 11 tasks in the OLMo 2 iteration of OLMES (OLMo et al., 2024; Gu et al., 2024b), to include 43 tasks across new families of capabilities. Here, we enumerate details from §3.3.

---

[50]https://huggingface.co/datasets/RJT1990/GeneralThoughtArchive
[51]https://huggingface.co/datasets/open-r1/OpenR1-Math-220k

Figure 29 Training curves of midtraining on canonical language model benchmarks (top), and our proposed base main task suites (bottom) for QA, Math and Code. We used the signal-to-noise ratio of early mid-training runs to make decisions about aggregating evaluation scores. Our resulting task averages had a better signal-to-noise ratio than individual benchmarks.

All task suites are publicly available at `github.com/allenai/olmes#olmo-3-eval-suite`.

Expanding OLMES tasks. We expanded our evaluation to target specific capabilities: new QA tasks focusing on science knowledge (SciQ, QASPER, SciRIFF), medical/lab knowledge (ProtocolQA, DBQA, MedMCQA, MedQA), math tasks (GSM Symbolic, Minerva MATH) and coding tasks (DS 1000, BigCodeBench, Deepseek LeetCode[52], MultiPL-E HumanEval, MultiPL-E MBPP). We use MultiPL-E to evaluate our multilingual code execution, limited to six core programming languages. Additionally, we track fill-in-the-middle (FIM) performance using HumanEval with the three settings from Bavarian et al. (2022): single-line infilling, multi-line infilling and random span infilling.

We support code execution in Python, C++, Java, JavaScript, PHP, Rust and Shell using AWS Lambda functions to grade instances in parallel, isolated environments of up to 50K generations simultaneously. In total, our environments graded 17.2 million generated code samples during OLMO 3 development, with up to 1.5K simultaneously. To ensure reproducibility, we release a lightweight Docker library for code execution without AWS infrastructure[53].

Additionally, OLMO 2 only tracked math and code capabilities after mid-training, as small models exhibit random-chance pass@1 performance on math and code tasks (Wei et al., 2022). Our base easy suite tracks perplexity over human-written math and code solutions (Huang et al., 2024b), which allows us to broadens the scope of capabilities we track during pre-training.

A.3.1 Base Evaluation Suites

Using the analysis tools described in the previous section, we construct two evaluation suite for decision making in pre-training: the **Base Easy** suite for small-scale data decisions and the **Base Main** suite for in-loop evaluation and mid-training data decisions. We kept the number of in-context examples and generation arguments consistent within each family of tasks, when possible.[54]

---

[52]We use 'Deepseek LeetCode' to refer to the 180 LeetCode problems used during development in Guo et al. (2024)

[53]Our code execution environments are publicly available at `github.com/allenai/olmes-docker`.

[54]We perform all evaluation using vLLM. To prevent performance discrepancies between versions, we pin to `v0.9.0.1` for evaluation during development, and pin to `v0.11.0` for all evaluation in the final report.

Figure 30 Scaling analysis for the OLMO 3 base evaluation suite. At the largest scale used to run from-scratch data ablations (grey line, a 1B model trained to 100B tokens), our 'base main' evaluation suite is too difficult to show improvement (top figures). Instead, we introduce a 'base easy' suite to compare models at small scales (bottom figures).

Figure 31 Relationship between bits-per-byte using the 'Easy' suite and final metrics using the 'Main' evaluation suite. We use the 'Easy' suite to make decisions at a small scale, which corresponds to an improvement at the large scale.

Table 43 describes the task configuration and metrics for the OLMO 3 base evaluation suite.

Base Easy Suite. For multiple-choice BPB, we simply use the correct answer as the continuation. For math BPB, we use the provided human-written solutions from Minerva MATH (Lewkowycz et al., 2022). For code BPB, we use the gold 'canontical' solution as provided in HumanEval and MBPP (Chen et al., 2021; Austin et al., 2021). For BPB over non-Python coding tasks, MultiPL-E did not release gold solutions (Cassano et al., 2022), so we generate silver continuations for 16 languages using `o4-mini-medium`[55]. Figure 30 shows the scaling behavior of the three base easy task clusters, where we see signal even at very small (190M parameter) model sizes.

One important property of the base eval suite is that a ranking of two small models on the base easy suite agrees with their ranking on the downstream base main suite. We validate this by measuring rank correlation between the easy and main task suites, as pictured in Figure 31.

---

[55]We release this generation set at `huggingface.co/datasets/allenai/multilingual_mbpp`

Base Main Suite. As a result of the clustering procedure, the base main suite tracks 6 task groups: MCQA STEM, MCQA Non-STEM, Gen, Math, Code, Code FIM. Unlike OLMo 2, we are tracking generative math and code tasks at pre-training. We chose to evaluate pass@k with the largest number of samples such that each task could evaluate on OLMo 2 7B on 1 H100 in under 30 minutes, in order to ensure the eval speed is not bottlenecked by any particular task. For tasks with a large enough n, we set $k = 16$ to match the GRPO group size, which we observed to act as an empirical upper-bound on the possible improvement from RL training. To decide on the the temperature and top-p, we ran a sweep and evaluated 5 models (OLMo 2 7B, 13B; Qwen 2.5 7B, 13B; Qwen 3 8B; Qwen et al., 2024; Yang et al., 2025a) to find an adequate configuration setting for high scores on both pass@1 and pass@k. Results are shown in Figure 32, and we select temperature and top-p of 0.6 for all base math and code evaluation.

Figure 32 To select generation arguments for base evaluation, we run a temperature and top-p sweep across 5 models. We use a reasonable configuration such that we can calculate both pass@1 and pass@k using the results of a single evaluation job.

Base Chat Suite. During mid-training, we refashion the chat eval suite (§4.1) for use evaluating base models, which served as a reference as to whether we expect our model to perform well after the adaptation pipeline. To do this, we used a standard, simple chat template (`Question: {text}\nAnswer:`) across all base models (both OLMo 3 and baseline models) and we included stop tokens to prevent degenerate responses. We also excluded tasks which required an API-based judge (AlpacaEval, SimpleQA) due to cost. In practice, we noticed most of the disagreements between the base main and base chat evaluation suites were due to noise, so we primarily used the base suite for making decisions.

Base Long-Context Suite. During the long-context extension phase, we evaluate long-context capability using RULER (Hsieh et al., 2024) as our primary development signal. As a complementary held-out set, we also use HELMET (Yen et al., 2025), noting that the HELMET *Recall* task directly implements several RULER evaluations (specifically, ruler-niah-mk-2, ruler-niah-mk-3, and ruler-niah-mv). Because we evaluate only base models at this stage, we disable chat templates within HELMET to ensure consistent scoring across models. For HELMET tasks requiring an LLM-as-a-judge, we use its default judge configuration (gpt-4o-2024-05-13). Taken together, RULER guides most model-selection decisions during long-context development, with HELMET providing an additional check on generalization.

Base Held-out Suite. We targeted one held-out evaluation task to match each family of capability: MMLU Pro for QA (Wang et al., 2024a), LBPP for code (Matton et al., 2024), Deepmind Math for math (Saxton et al., 2019), and BigBench Hard to measure broad coverage across unseen task types (Suzgun et al., 2022).

A.3.2 New Evaluation Benchmarks

Basic Skills. We developed a new benchmark, BASICSKILLS, to measure whether core capabilities are being acquired during pretraining. BASICSKILLS consists of 6 subtasks: basic arithmetic, string manipulation, simple coding, elementary logical reasoning, basic common sense, and simple pattern recognition. Each task isolates a single skill using a self-contained context that requires no external knowledge or additional information and can be completed through natural text continuation without relying on instruction-following abilities.

Gen2MC. One takeaway from OLMo 2 development was a sensitivity to task format. The clustering procedure furhter confirmed this, finding that generative scores rank models similarly as rank choice (RC) QA tasks, disagreeing with ranking of single-token multiple choice (MC) QA tasks (see Figure 5). In particular, the short-form generative QA tasks (GenQA in Table **??**) evaluate by comparing a generated answer to a bank of plausible answers, but these answer banks are often not complete, leading to false negatives. To address this,

	task	capability	ICL	metric	# sub
	Base Easy Suite				
	Minerva MATH★ (2022)	Math Gen	4^{α}	BPB	7
Code	HumanEval★ (2021)	Code Gen	3	BPB	-
	MBPP★ (2021)	Code Gen	3	BPB	-
	MT MBPP★ (2022)	Code Gen	3	BPB	17
QA	ARC (2018)	Science QA	5	BPB	2
	MMLU (2021b)	General QA	5	BPB	57
	CSQA (2019)	Commonsense QA	5	BPB	-
	HellaSwag (2019)	Language Modeling	5	BPB	-
	WinoGrande (2020)	Language Modeling	5	BPB	-
	SocialIQA (2019)	Social QA	5	BPB	-
	PiQA (2020)	Physical QA	5	BPB	-
	CoQA (2019)	Conversation QA	$0^{\dagger}$	BPB	-
	DROP (2019)	Passage QA	5	BPB	-
	Jeopardy (2024)	Trivia QA	5	BPB	-
	NaturalQs (2019)	General QA	5	BPB	-
	SQuAD (2016)	General QA	5	BPB	-
	SciQ★ (2017)	Science QA	5	BPB	-
	QASPER★ (2021)	Science QA	5	BPB	-
	Basic Skills★ (§A.3.2)	Basic QA	5	BPB	6
	DBQA★ (2024)	Science QA	5	BPB	-
	ProtocolQA★ (2024)	Science QA	5	BPB	-
	Lambada★ (2016)	Language Modeling	0	BPB	-
	MedMCQA★ (2022)	Medical QA	5	BPB	-
	MedQA★ (2021)	Medical QA	5	BPB	-
	SciRIFF★ (2024)	Science QA	5	BPB	-

Table 42 Details of the Olmo 3 base easy evaluation suite. Tasks were formatted as bits-per-byte (BPB) over the gold continuation, or rank choice (RC, following the setup in Gu et al. (2024b)). ★ = new additions to the base OLMo 2 suite (OLMo et al., 2024); † = few-shot examples are built-in the task; $^{\alpha}$ = human-written few-shot examples.

we introduce the **Gen2MC** benchmarks, which were constructed by taking the original question/answer pairs and generating incorrect multiple-choice distractor answers using a strong LLM. For each set of generated distractors, we manually review a set of 200 sample questions from the validation set before generating the full dataset. We create Gen2MC tasks for DROP, Jeopardy, NaturalQs, SQuAD, CoQA using GPT-4o for generating distractors, and fall-back to GPT-4.1 in cases where output parsing failed.

Masked Perplexity. We want our model to perform well on the diversity of requests from real user chat data; however, we don't want to overfit to the "style" of chat outputs. To avoid this, we use a simple token masking strategy, inspired by work in loss masking (Mindermann et al., 2022):

1. Fine-tune a 1B model on a tiny subset of the dataset (5%) with a small learning rate. The key idea is that we 'warm up' to the format of the target set without learning a lot of new knowledge.

2. Compute the token losses of the base model and the fine-tuned model on every sequence in the dataset and compute the difference: $\log p_{\text{SFT}}(y|x) - \log p_{\text{base}}(y|x)$

3. Mask tokens where the difference is greater than some threshold (found by inspection)

4. Also mask the user responses and tool calls (we don't want to model these for data selection) Use the loss at all the non-masked tokens positions for perplexity evaluations

In practice, we use Olmo 2 1B and the trained Olmo 2 1B SFT to compute the loss difference on target tokens. We use UltraChat and WildChat (Ding et al., 2023; Zhao et al., 2024a) as our masked perplexity sets.

	task	ICL	format	metric	temp	top-p	max toks	p@k (n)	# sub
	Base Main Suite								
Math	GSM8K★ (2021)	8^{α}	CoT EM	pass@k	0.6	0.6	512	1, 4 (8)	-
	GSM Symbolic★ (2024)	8^{α}	CoT EM	pass@k	0.6	0.6	512	1, 4 (8)	3
	Minerva MATH★ (2022)	4^{α}	CoT EM	pass@k	0.6	0.6	1024	1, 4 (4)	7
	MATH 500★ (2022; 2023)	4^{α}	CoT EM	pass@k	0.6	0.6	1024	1, 16 (32)	-
Code	HumanEval★ (2021)	3	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-
	MBPP★ (2021)	3	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-
	BigCodeBench★ (2024)	3	Code Exec	pass@k	0.6	0.6	1280	1 (5)	-
	DS 1000★ (2022)	3	Code Exec	pass@k	0.6	0.6	1024	1 (5)	-
	Deepseek LeetCode★ (2024)	0	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-
	MultiPL-E HumanEval★ (2022)	0	Code Exec	pass@k	0.6	0.6	1024	1, 16 (32)	6
	MultiPL-E MBPP★ (2022)	0	Code Exec	pass@k	0.6	0.6	1024	1, 16 (32)	6
FIM	HumEval FIM Single★ (2022)	0	FIM	pass@1	0.8	0.95	512	1 (10)	-
	HumEval FIM Random★ (2022)	0	FIM	pass@1	0.8	0.95	512	1 (5)	-
	HumEval FIM Multi★ (2022)	0	FIM	pass@1	0.8	0.95	512	1 (1)	-
STEM QA	ARC (2018)	5	MC	Acc	-	-	-	-	2
	MMLU STEM (2021b)	5	MC	Acc	-	-	-	-	19
	MedMCQA★ (2022)	5	MC	Acc	-	-	-	-	-
	MedQA★ (2021)	5	MC	Acc	-	-	-	-	-
	SciQ★ (2017)	5	MC	Acc	-	-	-	-	-
Non-STEM QA	MMLU Humanities (2021b)	5	MC	Acc	-	-	-	-	13
	MMLU Social Sci. (2021b)	5	MC	Acc	-	-	-	-	12
	MMLU Other (2021b)	5	MC	Acc	-	-	-	-	14
	CSQA (2019)	5	MC	Acc	-	-	-	-	-
	PiQA (2020)	5	MC	Acc	-	-	-	-	-
	SocialIQA (2019)	5	MC	Acc	-	-	-	-	-
	DROP Gen2MC★ (§A.3.2; 2019)	5	MC	Acc	-	-	-	-	-
	Jeopardy Gen2MC★ (§A.3.2; 2024)	5	MC	Acc	-	-	-	-	-
	NaturalQs Gen2MC★ (§A.3.2; 2019)	5	MC	Acc	-	-	-	-	-
	SQuAD Gen2MC★ (§A.3.2; 2016)	5	MC	Acc	-	-	-	-	-
	CoQA Gen2MC★ (§A.3.2; 2019)	$0^{\dagger}$	MC	Acc	-	-	-	-	-
	Basic Skills★ (§A.3.2)	5	MC	Acc	-	-	-	-	6
GenQA	HellaSwag (2019)	5	$RC_{\text{per-char}}$	Acc	-	-	-	-	-
	WinoGrande (2020)	5	RC_{none}	Acc	-	-	-	-	-
	Lambada (2016)	0	$RC_{\text{per-char}}$	Acc	-	-	-	-	-
	Basic Skills★ (§A.3.2)	5	$RC_{\text{per-token}}$	Acc	-	-	-	-	6
	DROP (2019)	5	GenQA	F1	0	1	100	-	-
	Jeopardy (2024)	5	GenQA	F1	0	1	50	-	-
	NaturalQs (2019)	5	GenQA	F1	0	1	50	-	-
	SQuAD (2016)	5	GenQA	F1	0	1	50	-	-
	CoQA (2019)	$0^{\dagger}$	GenQA	F1	0	1	50	-	-
	Base Held-out Suite								
	MMLU Pro (2024a)	5	MC	Acc	-	-	-	-	13
	LBPP★ (2024)	0	Code Exec	pass@k	0.6	0.6	4096	1 (32)	-
	Deepmind Math★ (2019)	5	CoT EM	pass@k	0.6	0.6	2048	1 (1)	-
	BigBench Hard (2022)	3	CoT EM	Acc	0.6	0.6	512	1 (1)	55

Table 43 Details of the OLMo 3 base evaluation suite. Tasks were formatted as multiple-choice (MC), rank choice (RC, following the setup in Gu et al. (2024b)), short-form generative (GenQA), chain-of-thought with exact-match scoring (CoT EM), code exeuction (Code Exec) or fill-in-the-middle coding (FIM). ★ = new additions to the base OLMo 2 suite (OLMo et al., 2024); † = few-shot examples are built-in the task; $^{\alpha}$ = human-written few-shot examples.

A.4 Post-Training Additional Training Details

A.4.1 Supervised Finetuning Details

Using Olmo Core infrastructure for SFT Training   Relative to pretraining, this involves a substantially smaller batch size, different data packing, and masking. This leads to an 8x faster training speed than open-instruct, dramatically improving our iteration speed. We use between 1 and 8 8xH100 nodes, or 1 to 4 8xB200 nodes to train our 7B reasoner and instruct models. We use 32 8xH100 nodes to train our 32B thinking model As a consequence of using olmo-core, our batch size is now measured in tokens instead of instances, and we train with document packing instead of padding. We train all of our 7B SFT models with a batch size of 1m tokens and 32B SFT models with a batch size of 4m tokens, for two epochs, with packing, and a 32,768 sequence length. Our hyperparameter settings are also summarized in Table 44.

	7B Thinking SFT	32B Thinking SFT	7B Instruct SFT
Total Tokens	45.4B	45.2B	3.4M
Learning Rate	5.0×10^{-5}	1.0×10^{-4} souped with 5.0×10^{-5}	8.0×10^{-5}
Num. GPUs	64	256	8-64
Max Sequence Length	32K	32K	32K

Table 44 **Training hyperparameters for OLMO 3 THINK SFT and OLMO 3 INSTRUCT SFT.** All GPUs used were H100s.

A.4.2 Preference Tuning Details

Training Settings   Given a preference dataset $\mathcal{D} = \{(x, y_c, y_r)\}$ of prompts x and corresponding chosen and rejected responses $y_c > y_r$, we optimize the model policy π_θ on a length-normalized DPO loss (Lambert et al., 2024):

$$\max_{\pi_\theta} \mathbb{E}_{(x,y_c,y_r)\sim\mathcal{D}} \left[\log \sigma \left(\frac{\beta}{|y_c|} \log \frac{\pi_\theta(y_c|x)}{\pi_{\mathrm{ref}}(y_c|x)} - \frac{\beta}{|y_r|} \log \frac{\pi_\theta(y_r|x)}{\pi_{\mathrm{ref}}(y_r|x)} \right) \right]$$

where π_{ref} is the initial reference policy and β is a hyperparameter that regularizes learning via an implicit Kullback-Lieber (KL) divergence penalty between the reference policy and the training policy.

We sweep learning rate and preference dataset size, as we observe that performance increases up until some task-dependent optimal optimization point beyond which further tuning hurts (Figure 23). All other hyperparmeters are kept fixed. See Table 45 for exact hyperparameters. We train our 7B models using 2-4 8xH100 nodes, and our 32B models with 8-16 8xH100 nodes.

	7B Thinking DPO	32B Thinking DPO	7B Instruct DPO
Num. Preference Pairs	150k	200k	260k
Num. Epochs	1	1	1
DPO β	5	5	5
Learning Rate	8.0×10^{-8}	7.0×10^{-8}	1.0×10^{-6}
LR Schedule	Linear decay	Linear decay	Linear decay
Warmup Ratio	0.1	0.1	0.1
Num. GPUs	32	64-128	16
Batch Size	128	128	128
Num. GPUs	32	64-128	16
Max Sequence Length	16K	8K	16K

Table 45 **Training hyperparameters for OLMO 3 THINK DPO and OLMO 3 INSTRUCT DPO.** All GPUs used were H100s.

DPO prompt mixing

Name	Avg.	Subset of Olmo 3 Instruct Benchmarks								
		MMLU	BBH	GPQA	AGI	MATH	CHE	LCB	IFEval	AE2
Development SFT	50.1	66.3	44.2	29.9	58.6	56.2	70.0	13.8	**82.1**	29.8
Base mix (uniform* sample)	54.3	**68.1**	48.1	32.1	62.7	67.3	**68.5**	17.0	79.3	**45.4**
Ablate code	53.6	64.7	**51.6**	33.0	**65.2**	**67.9**	65.9	17.7	75.8	40.6
Ablate math	**54.4**	67.8	49.2	33.0	64.8	67.2	67.0	20.4	77.3	42.9
Ablate science	52.8	66.4	49.9	31.7	64.2	67.0	60.0	19.8	76.3	39.6
Ablate chat	53.1	67.1	51.3	30.6	64.8	67.6	59.3	**21.2**	76.3	39.3
Ablate inst. following	50.3	66.1	51.0	29.5	62.5	66.3	48.3	18.7	75.2	34.8
Ablate safety	51.0	66.3	48.6	**34.2**	63.5	67.3	51.0	18.1	74.7	35.4
Ablate misc/SFT unused	48.3	66.6	49.9	29.7	64.2	65.3	38.6	14.9	74.1	31.2
Upsample code	51.1	67.7	48.6	31.7	63.8	65.9	51.7	18.0	76.0	36.3
Upsample math	53.3	67.5	48.6	29.5	62.3	66.4	66.7	17.5	78.4	42.6
Upsample chat	53.0	67.0	46.8	30.6	61.6	65.7	68.3	15.6	76.9	44.7

Table 46 Development results for DPO prompt domain mixing (rounded to one decimal place). Overall, we find that (1) all prompt domains are useful for performant tuning, but (2) the exact ratios for each domain are challenging to ascertain systematically; the prompt domain does not necessarily correspond directly to the domains in which performance improves.

Model pool for LLM-judged pairs For our Instruct model, we generate completions on our prompt pool with the following models: GPT-OSS 20B, GPT-OSS-120B (Agarwal et al., 2025), GPT-4.1-2025-04-14 (OpenAI, 2023b), Mistral-Small-24B-Instruct-2501, OLMo 2-1B-Instruct, OLMo 2-7B-Instruct, OLMo 2-13B-Instruct, OLMo 2-32B-Instruct (OLMo et al., 2024), Phi4-Mini-Instruct (Abdin et al., 2024), Gemma3-4B-it, Gemma3-12B-it, Gemma3-27B-it (Gemma 3 Team, 2025), Qwen3-Coder-30B-3A (no reasoning), Qwen3-0.6B (no reasoning), Qwen3-1.7B (no reasoning), Qwen3-4B (no reasoning), Qwen3-8B (no reasoning), Qwen3-14B (no reasoning), Qwen3-32B (no reasoning), Qwen3-30B-3A (no reasoning) (Yang et al., 2025a), QwQ-32b (Qwen Team, 2025), Yi-9B, Yi-34B (Young et al., 2024).

For each prompt, we sample four model completions and judge them via a GPT-4.1 judge with the Ultrafeedback judge prompts (Lambert et al., 2024; Cui et al.).[56] To enforce a meaningful delta between chosen and rejected responses, we enforce our judge pipeline to sample responses from exactly two of the following smaller and/or previous generation models which show lower overall performance: OLMo 2-1B-Instruct, OLMo 2-7B-Instruct, Yi-9B, Yi-34B, Phi4-Mini-Instruct, Qwen3-0.6B (no reasoning), Qwen3-1.7B (no reasoning). Without this intervention, we would have a 33% chance of sampling at least 2 weak models out of our 4 samples from our model pool for judgment, providing limited contrast in preference pairs. We binarize into preference pairs by selecting the *worst* response out of the four to be rejected, and the best as chosen.

A.4.3 Reinforcement Learning Details

We provide full training curves for our 7B reasoner in Figure 35. The overall reward increases steadily over training. The KL divergence grows gradually and reflects stronger deviation from the reference policy. The response length becomes longer and stabilizes at a higher level. Domain-specific verifier rewards display consistent gains in math and moderate fluctuations in code. The IfEval reward rises throughout training. The two general-quality verifiers also show clear and sustained improvement. Together, these trends indicate that the policy improves both specialized skills and overall response quality. The full hyperparameters for all RL experiments are provided in in Table 47.

A.4.4 RL-Zero Details

We detail the prompt used for math in Figure 33. Prompts of other domains are quite similar, see the open-instruct codebase for details.

RL-Zero Math Prompt

Solve the following math problem step by step.
The last line of your response should be the answer to the problem in form Answer: $Answer (without quotes) where $Answer is the answer to the problem.

{Math Question}

Remember to put your answer on its own line after "Answer:"

Figure 33 RL-Zero Prompt for Math Task.

A.5 Post-Training Additional Data Details

A.5.1 Filtering for Dolci Think-SFT

In this section we detail the filtering methods created primarily for training OLMo 3 THINK, which was also used for mid-training and OLMo 3 INSTRUCT data. Each phase of filtering would remove 0-1% of data across most available or generated reasoning traces. Some data, such as Nvidia's Nemotron Post-training datasets (Nathawani et al., 2025) had very few samples removed relative to their peers.

---

[56]We ran initial experiments employing a GPT-5 judge, but results suggested that the GPT-4.1 judge actually performed better.

Figure 34 LLM Judge Prompt for Non-verifiable Tasks.

1. **Source Filtering** We perform some filtering to remove non-compliant licenses or data that will not be useful. E.g. for GeneralThoughts traces used in mid-training, we filtered to only commercially friendly licensed prompts. For OpenThoughts2, we removed ShareGPT prompts due to questionable provenance (as done in Tulu 3). For LlamaNemotron Post-Training we filter to only reasoning samples from DeepSeek and Qwen that have not been touched by Llama models.

2. **Format Filtering** We remove truncated answers (i.e. if they have <think> and no </think>) and empty outputs (empty responses). https://github.com/allenai/open-instruct/blob/7ba4cd0/scripts/data/filtering_and_updates/filter_cots.py

3. **Domain Specific Accuracy Filtering** We check accuracy for many domains, such as precise instruction following, code, or math. Additionally, for chat domains we use included metadata in some datasets such as Wildchat to remove responses or prompts tagged as unsafe. https://github.com/allenai/open-instruct/blob/7ba4cd0/scripts/data/filtering_and_updates/filter_wildchat.py

4. **General Content Filters** Here we remove mention of date cutoffs to try and avoid hallucinations of model characteristics and any mention in the user prompt or completion that indicates the date is to or from any model. Maintaining identity of models trained on heavily distilled data takes a meaningful amount of data work and system prompt design. https://github.com/allenai/open-instruct/blob/7ba4cd0/scripts/data/filtering_and_updates/filter_datasets_sequential.sh

Figure 35 Reward, KL, response length, and per-verifier reward over the final RL run for OLMo 3 Thinker.

	7B Thinking RL	32B Thinking RL	7B Instruct RL	7B RL-Zero Math
Dataset Size	104,869	104,869	171,950	13,314
Learning Rate	1.0×10^{-6}	2.0×10^{-6}	1.0×10^{-6}	1.0×10^{-6}
Minibatches	1	1	4	1
LR Schedule	Constant	Constant	Constant	Constant
Training Steps	1,400	750	450	2,000
Max Prompt Length	2,048	2,048	2,048	2,048
Response Length	32,768	32,768	8,192	16,384
Unique Prompts Per Batch	64	128	64	32
Group Size	8	8	8	8
TIS Cap	-	2.0	0.2	2.0
Sampling Temperature	1.0	1.0	1.0	1.0
Clip-Lower	0.2	0.2	0.2	0.2
Clip-Higher	0.272	0.272	0.272	0.272
Num Learner GPUs	16	64	8	8
Num Actor GPUs	56	160	56	64
GPUs per Actor (TP)	1	8	1	1
Max Asynchrony	1	8	8	8

Table 47 RL training hyperparameters for Olmo 3 Think, Olmo 3 Instruct and OLMO RL0. All GPUs used were H100s.

5. **Repetition Filtering** Many open weights reasoning models have tendencies to perform extreme repetitions, even in thinking traces that result in a correct answer. In particular, we find that .1% of responses from QwQ have mass repetition. We filter this roughly by searching for heavily repeated (10x+) sentences, paragraphs, or (50x+) phrases. https://github.com/allenai/open-instruct/blob/7ba4cd0/scripts/data/filtering_and_updates/filter_ngram_repetitions.py

6. **Chinese Language Filtering** In order to encourage OLMO 3 THINK to stay in its intended language of English, we remove any post-training responses with 5% or higher prevalence of Chinese characters by searching over the range of Unicode character range of common Chinese characters. https://github.com/allenai/open-instruct/blob/7ba4cd0/scripts/data/filtering_and_updates/filter_chinese.py

A.5.2 Tool-use data

Additional details about the Science QA dataset Citation graph-based queries are produced by prompting GPT-5 in a few-shot setup to create query templates, e.g., `What are the top-three most cited papers by {AUTHOR} on {TOPIC}?` which are subsequently instantiated with real paper entities. Content-based questions are generated by a GPT-5-based agent equipped with the ASC server, which retrieves relevant papers and formulates grounded questions that can be answered using retrieved text. For both types of queries, to obtain corresponding tool-use trajectories we employ a GPT-4.1-mini agent with access to the same ASC server. All tool call outputs are derived from actual environment responses rather than synthetic completions.

Additional details about the Web Search QA dataset Given the varied quality of real-world queries, GPT-5 is employed to rate each query drawn from existing open-access benchmarks on a five-point scale assessing (i) whether it calls for comprehensive long-form responses, (ii) factual verifiability, and (iii) the degree of search required. Only queries scoring 4 or 5 on these criteria are retained. We then use an agent equipped with web search and browsing via the Serper API, and scientific snippet retrieval via ASC to generate tool-use trajectories for these queries. This agent is instructed with tool specifications and step-by-step search instructions, resulting in detailed trajectories containing both tool calls and environment outputs. We then filter out trajectories that yield incorrect answers (where ground truth is available), and only keep trajectories

that adhere to the expected output format. Additionally, since the environment outputs for the webpage fetching tool of the Serper API are quite long (typically entire webpages), we used GPT 5 to summarize the content of the web pages and only retained the summaries in the training data.

Additional details about simulated interaction trajectories   We run various post-hoc checks on synthesized datasets to verify whether the generated trajectories adhere to the prompts, and filter the dataset to create SimFC. We filtered out trajectories where the function calls included functions not part of the presented APIs. Our data synthesis prompts explicitly target multi-turn, multi-step, parallel function calls (i.e., multiple calls per assistant turn), and refusals, ande filter out the trajectories that do not conform to such requirements specified in the prompts.

Prompt for Generating Multi-Turn Function-calling Interactions

You are provided an API with the details of the functions shown in a JSON format. Use this API to write a simulated interaction between a user, an assistant that can call the functions in the API, and the environment. The interaction should refer to three roles: `"user"`, `"assistant"`, and `"environment"`. Their messages should be represented as Python dicts with `"role"` and `"content"` fields.

If the assistant is making function calls, they should be shown under a `"function_calls"` field instead of the `"content"` field. The interaction should start with a user request, contain multiple steps of the assistant making function calls while interacting with the user for additional inputs, and should conclude with the assistant performing the user's requested action. Please generate a simulated interaction with at least 5 function calls. Ensure that at the end of each turn, the assistant should address the request of the user by creating an assistant message with a text in the `"content"` field.

Here is an example:

```
API:
[
  {"name": "get_borrowed_books", "description": "Get borrowed books by user ID",
    "parameters": {"user_id": {"type": "int"}}},
  {"name": "get_user_info", "description": "Get user information",
    "parameters": {"prefix": {"type": "str", "required": false},
        "email": {"type": "str", "required": false}}},
  {"name": "get_late_fines", "description": ...}
]

INTERACTION:
[
  {"role": "user", "content": "How many users with the name Yoda exist?"},
  {"role": "assistant", "function_calls": "get_user_info(prefix='Yoda')"},
  {"role": "environment", "content": "{\"results\": [{\"id\": 23}]}"},
  {"role": "assistant", "content": "There is one user with that name."},
  {"role": "user", "content": "How many books have they borrowed?"},

  ... additional turns ...

  {"role": "assistant", "content": "Luke Skywalker has borrowed one book."}
]
```

Here is the real task:

```
API: {}
INTERACTION:
```

Figure 36 Illustrative Prompt for generating multi-turn function-calling interactions with simulated environment feedback (prompt has been truncated for readability).

A.5.3   Coding Data Synthesis Pipeline

To construct reinforcement learning (RL) data for code, we required pairs of *(problem, test cases)*. We curate a diverse set of prompts for coding problems, including AceCoder (Zeng et al., 2025a), Klear-Reasoner Code (Su et al., 2025c), Nemotron Post-training Code (NVIDIA AI, 2025), SYNTHETIC-2 code (PrimeIntellect, 2025),

Figure 37 Illustrative Prompt for generating function-calling refusals i.e. when the task is not feasible given the available functions (prompt has been truncated for readability).

Open-Code Reasoner (Ahmad et al., 2025). We use the klear-reasoner and SYNTHETIC-2 test cases directly. For the other datasets, we run prompts through the following synthetic data pipeline:

- **Problem rewriting.** Given a coding problem, we first prompted GPT-4.1 to rewrite the description so that it either (a) included a function signature, or (b) explicitly specified that the solution should read from and write to standard input/output (stdio)

- **Solution generation.** GPT-4.1 was then prompted to provide a corresponding solution. Depending on the problem type, this was either a Python function matching the given signature, or a program reading from and writing to stdio. When the original problem source included a reference solution, we included it in the prompt

- **Test case generation.** GPT-4.1 was further prompted to generate test cases in the appropriate format

Dataset	Original Size	Format Filtering	Domain Filtering	General Filtering	Content Filtering	Repetition Filtering	Chinese Filtering	Final Size
WildChat (Tülu 3)	57,407	1.61%	14.57%	0.75%	3.10%	–	1.09%	45,917
WildChat (New)	74,997	1.53%	48.09%	0.80%	3.13%	0.02%	1.16%	36,417
OpenAssistant1	7,094	0.08%	–	0.22%	–	–	3.86%	6,800
OpenThoughts3-Regen	1,200,000	3.22%	–	0.00%	–	< 0.01%	0.04%	1,160,972
Persona Precise IF	224,448	0.19%	–	0.03%	0.29%	< 0.01%	0.08%	223,123
Val Precise IF (QwQ)	286,003	–	–	–	0.62%	< 0.01%	1.17%	135,851
Synthetic-2-SFT-Verified	104,913	0.01%	–	0.06%	–	< 0.01%	0.32%	104,569
Saurabh Code Mix	884,767	–	–	–	–	< 0.01%	< 0.01%	884,570
CoCoNot	10,460	0.57%	–	1.57%	–	–	0.10%	10,227
WildGuard	38,794	0.37%	–	1.17%	0.54%	< 0.01%	0.12%	38,315
WildJailbreak	41,420	0.13%	–	0.21%	0.61%	–	< 0.01%	41,100
Aya	98,863	0.15%	–	1.70%	–	< 0.01%	5.62%	98,598
TableGPT	4,982	0.02%	–	0.00%	–	–	0.06%	4,981

Table 48 Filtering statistics showing percentage of prompts removed at each major filtering stage for reasoning datasets. "–" indicates filtering was not applicable or no samples were removed.

(function-based or stdio-based)

A.6 Post-Training Additional Evaluation Details

A.6.1 General Evaluation Settings

For post-training, we focus exclusively on generative evaluations, in which we generate completions until a max length is reached or eos token is generated (as opposed to multiple-choice-based evaluations used in pretraining), better matching real-world downstream usage.

Following DeepSeek R1 report (Guo et al., 2025) and Nvidia Nemotron (Adler et al., 2024) we use a sampling temperature of 0.6 and top-p of 0.95.[57] We strip thinking traces from the answer text when generated. We account for the variance this induces in smaller benchmarks (e.g. AIME, which is made up of 30 questions) by taking multiple samples and reporting the overall average performance. For QA tasks (e.g. BBH, MMLU), we create a unified set of 'Olmo 3' regexes for answer extraction, covering a wide variety of potential answer tempaltes. We additionally update AlpacaEval 2 Length Controlled (LC) (Dubois et al., 2024) to use GPT-4.1 as a judge instead of the original GPT-4-Turbo (OpenAI, 2023b) both to increase the reliability of the evaluation and to save ~90% of inference costs. Importantly, our evaluation settings are **unified across thinker and instruct models**, simplifying our evaluation development process.

A.6.2 Safety Evaluations Overview

The safety evaluations that were tested upon during training runs and whose average was reported earlier were the same set from OLMo2 (OLMo et al., 2024) and Tulu3 (Lambert et al., 2024). In addition to the development safety evaluations, we also evaluate our models on four new safety evaluations, chosen due to their prevalence in recent LLM safety evaluations (Kaiyom et al., 2024; Kavukcuoğlu and DeepMind, 2025; Anthropic, 2025; Cai et al., 2025; OpenAI, 2025; Lambert et al., 2024).

Development safety evaluations include HarmBench (Mazeika et al., 2024), DoAnythingNow (DAN; Shen et al., 2024), XSTest (Röttger et al., 2023), WildGuard-Test (Han et al., 2024), WildJailbreak-Test (Jiang et al., 2024), and TrustLLM-JailbreakTrigger (Huang et al., 2024a).

---

[57]We find that both thinking models degenerate quickly when evaluated with low temperatures (as used in OLMo 2), while instruction models can be evaluated at this higher temperature.

Unseen safety evaluations that we added were Toxigen (Hartvigsen et al., 2022), StrongReject (Souly et al., 2024), Weapons of Mass Destruction Proxy (WMDP; Li et al., 2024b), and Bias Benchmark for QA (BBQ; Parrish et al., 2022).

Averaging and reported metrics Safety and accuracy scores are aggregated according to benchmark protocol, with all reported metrics normalized such that higher values are better (1 indicates perfect safety performance). Specifically, we report the average of: *refusal accuracy*, i.e., inverted ASR (Attack Success Rate), for DoAnythingNow, Harmbench, Wildguard, TrustLLM-JailbreakTrigger, Toxigen, and StrongReject; *accuracy* for XSTest and BBQ; the average of inverted ASR for Wildjailbreak (harmful) and ASR for Wildjailbreak (benign); and inverted accuracy (i.e., error rate) for WMDP. For the safety benchmarks, models were evaluated with a top-p of 0.95 and sampling temperature of 0.7.

We explain all of the evaluations in more detail below:

- **HarmBench (Mazeika et al., 2024)** evaluates models' refusal to comply with a diverse suite of harmful prompts, distributed across both functional and semantic categories. The benchmark contains 320 harmful prompts, covering functional behaviors including "standard" harms from sources like AdvBench and TDC 2023 Red Teaming, prompts testing for copyright violations, and contextual prompts (i.e., prompts combining an instruction with a context such as a document).[58] The semantic category includes prompts representing seven types of real-world harms: cybercrime, unauthorized intrusion, chemical/biological/drug-related, copyright infringement, misinformation, harassment, illegal activities, and general harm. Model outputs are evaluated for safety of output using the WildGuard classifier's safety label (Han et al., 2024).

- **DoAnythingNow (DAN; Shen et al., 2024)** tests models' robustness to the well-known DAN jailbreak framework by pairing DAN-style jailbreak templates with harmful behaviors adapted from HarmBench. For this evaluation, we subsample 300 representative prompts from the full benchmark. Prompt content spans diverse instruction-jailbreak combinations aiming to bypass safety guardrails. We compute refusal accuracy[59] using the WildGuard classifier's refusal label (Han et al., 2024) to assess whether model outputs refuse or comply with the harmful instructions.

- **XSTest (Röttger et al., 2023)** measures models' *over-refusal* tendencies, i.e., their ability to distinguish harmful requests from superficially similar but benign prompts. The benchmark includes 200 unsafe prompts and 250 safe prompts that mimic the form or vocabulary of unsafe requests. Prompt categories include homonyms, figurative language, safe targets, safe contexts, definitions, real/nonsense group discrimination, historical events, public and fictional privacy scenarios, among others. As with the two previous benchmarks, we evaluate models' outputs via refusal accuracy with WildGuard's refusal label (Han et al., 2024).

- **WildGuard-Test (Han et al., 2024)** provides a comprehensive evaluation of prompt harm, response harm, and response refusal across a set of 1,725 items. Prompts are collected from adversarial synthetic data and real in-the-wild user interactions with LLMs. We evaluate on the subset of 749 adversarial prompts. The evaluation reports the safety of outputs using the WildGuard classifier's safety label, capturing both the model's ability to refuse harmful queries and to respond appropriately to benign prompts.

- **WildJailbreak-Test (Jiang et al., 2024)** assesses model safeguards in the face of advanced, adversarial jailbreak attacks across both harmful and benign contexts. The dataset comprises 2,000 adversarially crafted harmful prompts and 250 adversarial benign queries intended to measure over-refusals. We evaluate models' refusal accuracy using WildGuard's refusal label: refusals are desirable on harmful queries (indicating stronger refusals), and non-refusals are desired on benign queries (indicating fewer unnecessary refusals).

- **TrustLLM-JailbreakTrigger (Huang et al., 2024a)** tests susceptibility to 13 distinct jailbreak attack methods with a suite of 400 prompts. Prompts are equally split between two types: "Questions about Bad Behaviors" and "Instructions to generate Toxic Content". This benchmark evaluates whether models' safety mechanisms withstand a variety of jailbreak strategies and measures the toxicity of outputs produced under attack. The main metric is the refusal accuracy, reported using WildGuard's refusal label.

- **Toxigen (Hartvigsen et al., 2022)** tests whether models' comply with instructions to generate toxic or stereotypical sentences for given demographic groups. Each prompt includes a list of 10 examples of toxic

---

[58] In our implementation, the context string is merged into the input prompt for contextual examples.

[59] Refusal accuracy corresponds to 1-ASR (attack success rate).

sentences, to which an unsafe model would add new toxic sentences. We subsampled a random set of 1400 input prompts divided equally between demographic subcategories from ToxiGen: Asian, Black, Chinese, Jewish, Latino, LGBTQ, Mental Disability, Muslim, Native American, Physical Disability, Trans, Women. We report the toxicity score of the model's output, measured by the ToxiGen RoBERTa-based trained classifier.[60]

- **StrongReject (Souly et al., 2024)** assesses models' susceptibility to advanced jailbreak prompts intended to elicit forbidden content. Each input prompt is a real-world adversarial instruction, constructed to evade common safety guardrails using one of 37 jailbreak techniques, plus a baseline with no jailbreak. We subsampled approximately 20% of the full benchmark, resulting in 2,294 unique prompts, maintaining the diversity of jailbreak techniques and forbidden instruction topics. The model's output is scored on a 1-5 safety scale according to the risk or compliance with the forbidden prompt, using a classifier that is uses a version of the StrongReject classifier[61] that is a LoRa adapter merged with the base Gemma-2B (Gemma Team et al., 2024) model by the benchmark creators. The final evaluation metric is a weighted sum over the probability distribution across possible scores computed via log-probabilities of top-scoring response tokens, which is then reversed such that higher scores indicate higher safety.

- **Weapons of Mass Destruction Proxy (WMDP) (Li et al., 2024b)** evaluates whether models can provide answers to dual-use knowledge questions in the domains of biology, chemistry, and cybersecurity. Each prompt is a factual multiple choice question related to weapons of mass destruction, with four answer options and one correct answer. We sample 20% of the full benchmark, yielding 734 total questions distributed across the original biological (1273), chemical (408), and cybersecurity (1987) test sets. Prompts include the instruction we use when evaluating MMLU (Wang et al., 2024a):

  > The following are multiple choice questions. Summarize your reasoning concisely, then conclude with 'Therefore, the answer is: X' where X is one of A, B, C, or D.\n \n

Models' outputs are evaluated via exact match (based on regex parsing) where only outputs with a correctly formatted, unambiguous final answer string are considered correct.

- **Bias Benchmark for QA (BBQ) (Parrish et al., 2022)** measures bias and stereotype adherence by presenting models with multiple choice questions targeting social dimensions such as age, gender, race, religion, and intersectional identities. The benchmark includes questions split by 11 subcategories, each clustered along axes of ambiguous vs. unambiguous and stereotypical vs. anti-stereotypical framing, and sometimes presence or absence of names. For our evaluation, we drew a subset of ~500 questions per subcategory (excluding intersectional combinations), distributed evenly across prompt types (ambiguous/unambiguous, stereotypical/anti-stereotypical, and, with or without names), resulting in 4482 total instances. Each prompt is presented in the same structured format as WMDP.[62] Model responses are evaluated for *accuracy* (proportion of correct answers) and for *bias*, using a regex-based string parser (similar to BBQ). Accuracy simply measures whether models picked the right answer. Bias is quantified according to the protocol in Parrish et al. (2022): ambiguous and disambiguated bias scores are computed as the frequency with which non-unknown outputs reinforce stereotypes within each prompt type (e.g., the model incorrectly picks the stereotypical answer).

---

[60] https://huggingface.co/tomh/toxigen_roberta

[61] https://huggingface.co/qylu4156/strongreject-15k-v1

[62] Note that this is different from the more restrictive HELM-Safety prompting format Kaiyom et al. (2024) which only scores based on the first generated token.

Skill	Benchmark	Olmo 3 Think 7B	Olmo 3 Think 7B-DPO	Olmo 3 Think 7B-SFT	OpenThinker3-7B	Nemotron-Nano-9B-v2	DeepSeek-R1-Distill-Qwen-7B	Qwen 3 8B (with reasoning)	Qwen 3 VL 8B Thinker	OpenReasoning Nemotron 7B
DoAnythingNow		23.4	19.6	19.3	1.2	56.7	34.3	53.1	83.0	2.3
HarmBench		75.4	72.7	67.8	26.6	69.4	50.7	74.0	81.9	20.0
TrustLLM-JailbreakTrigger		72.0	65.2	64.8	2.9	62.6	50.1	56.7	77.0	6.9
WildJailbreak-Test	Harmful	39.0	27.5	23.4	0.2	28.7	4.5	12.3	38.6	0.5
WildJailbreak-Test	Benign	98.8	98.5	99.1	98.8	97.3	98.0	99.7	98.0	97.1
WildGuard-Test		93.8	93.9	90.2	48.1	88.4	69.2	82.9	93.0	42.6
XSTest		90.9	91.6	91.6	59.5	92.5	68.4	87.2	94.2	61.0
BBQ	Accuracy	89.2	84.8	86.6	80.3	92.0	78.0	91.8	86.6	82.6
BBQ	Bias - Ambig.	6.5	8.4	7.3	11.4	5.8	9.4	5.5	8.9	7.1
BBQ	Bias - Disambig.	1.7	1.1	1.7	2.2	0.7	2.4	1.5	1.0	2.3
StrongReject		79.0	75.5	74.8	56.7	85.6	72.4	73.4	82.8	58.3
Toxigen		100.0	99.9	100.0	97.6	100.0	99.7	100.0	99.9	86.4
WMDP		42.7	43.4	46.4	43.8	38.3	55.9	34.9	38.7	51.8

Table 49 Olmo 3 Think 7B and comparisons on the safety benchmarks.

Skill	Benchmark	Olmo 3 Instruct 7B	Olmo 3 Instruct 7B-DPO	Olmo 3 Instruct 7B-SFT	Qwen 3 8B (without reasoning)	Qwen 3 VL 8B Instruct	Qwen 2.5 7B	Olmo 2 7B Instruct	Apertus 8B Instruct	Granite 3.3 8B Instruct
DoAnythingNow		75.8	83.9	88.3	81.0	66.9	60.2	96.7	43.0	36.8
HarmBench		94.1	94.9	87.0	72.7	84.3	79.0	89.9	81.0	85.1
TrustLLM-JailbreakTrigger		78.3	85.9	87.0	76.1	78.7	63.4	94.8	58.0	62.3
WildJailbreak-Test	Harmful	69.4	72.7	80.4	21.2	37.5	13.4	84.6	43.7	66.9
WildJailbreak-Test	Benign	97.5	96.5	88.8	99.5	98.3	99.5	88.3	95.2	82.9
WildGuard-Test		99.6	99.7	98.9	86.9	89.9	88.0	99.3	91.0	94.0
XSTest		92.7	92.4	89.3	91.1	93.5	93.2	91.2	90.5	89.3
BBQ	Accuracy	79.0	75.1	74.3	87.0	87.8	88.3	70.7	73.8	68.3
BBQ	Bias - Ambig.	8.7	9.8	8.3	8.5	8.1	7.1	8.8	7.3	3.2
BBQ	Bias - Disambig	2.6	3.0	2.7	2.0	-0.2	3.4	1.8	2.9	2.8
StrongReject		88.1	88.9	94.1	85.8	89.7	78.4	95.3	78.4	82.5
Toxigen		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
WMDP		46.7	48.6	48.4	36.8	38.7	40.6	52.3	47.8	46.0

Table 50 Olmo 3 Instruct 7B and comparisons on the safety benchmarks.

Skill	Benchmark	Olmo 3 Think 32B	Olmo 3 Think 32B DPO	Olmo 3 Think 32B SFT	Qwen 3 32B	Qwen 3 VL 32B Thinking	Qwen 2.5 32B Instruct	Gemma 3 27B Instruct	Gemma 2 27B Instruct	Olmo 2 32B Instruct	DeepSeek-R1-Distill-Qwen-32B
DoAnythingNow		20.2	15.6	16.7	59.0	88.7	74.2	31.0	30.1	73.7	46.0
HarmBench		73.5	69.7	66.5	67.3	75.2	87.1	67.8	90.1	87.2	64.0
TrustLLM-JailbreakTrigger		73.3	69.6	68.3	60.7	75.6	83.8	73.9	74.8	77.3	55.3
WildJailbreak-Test	Harmful	25.6	17.5	17.6	12.6	47.0	22.3	17.3	39.7	49.8	13.7
WildJailbreak-Test	Benign	99.7	99.6	99.2	100.0	94.0	99.7	100.0	98.5	99.2	99.2
WildGuard-Test		89.4	86.5	86.3	81.3	92.9	91.9	88.7	92.0	98.4	81.7
XSTest		93.9	92.1	93.0	89.6	93.4	93.3	91.6	89.8	92.0	78.1
BBQ	Accuracy	88.2	88.5	90.6	89.7	90.5	91.1	83.8	86.3	84.6	88.1
BBQ	Bias - Ambig.	9.2	8.2	6.9	7.1	5.6	7.7	11.2	10.5	9.6	8.1
BBQ	Bias - Disambig.	1.1	0.2	0.8	0.1	0.0	0.2	0.9	0.4	1.2	0.4
StrongReject		80.8	77.2	75.9	79.3	88.5	85.4	84.5	88.2	87.7	79.1
Toxigen		100.0	100.0	100.0	100.0	99.9	100.0	99.5	100.0	100.0	100.0
WMDP		34.8	34.9	40.2	24.0	31.0	32.4	38.3	39.6	41.9	30.9

Table 51 Olmo 3 Think 32B and comparisons on the safety benchmarks.