INTELLECT-1 Technical Report

Sami Jaghouar
Prime Intellect

Jack Min Ong
Prime Intellect

Manveer Basra
Prime Intellect

Fares Obeid
Prime Intellect

Jannik Straube
Prime Intellect

Michael Keiblinger
Prime Intellect

Elie Bakouch
Hugging Face

Lucas Atkins
Arcee AI

Maziyar Panahi
Arcee AI

Charles Goddard
Arcee AI

Max Ryabinin
Together AI

Johannes Hagemann
Prime Intellect
johannes@primeintellect.ai

Abstract

In this report, we introduce INTELLECT-1, the first 10 billion parameter language model collaboratively trained across the globe, demonstrating that large-scale model training is no longer confined to large corporations but can be achieved through a distributed, community-driven approach.

INTELLECT-1 was trained on 1 trillion tokens using up to 14 concurrent nodes distributed across 3 continents, with contributions from 30 independent compute providers dynamically joining and leaving the training process, while maintaining 83-96% compute utilization and 36.2–41.4% model FLOPS utilization.

We leverage PRIME, our scalable distributed training framework designed for fault-tolerant, high-performance training on unreliable, globally distributed nodes. Key innovations in PRIME include the `ElasticDeviceMesh`, which manages dynamic global process groups for fault-tolerant communication across the internet and local process groups for communication within a node, live checkpoint recovery, kernels, and a hybrid DiLoCo-FSDP2 implementation.

Using PRIME with DiLoCo and our custom int8 all-reduce, we achieve a $400\times$ reduction in communication bandwidth compared to traditional data-parallel training settings while delivering comparable performance.

These results demonstrate the feasibility and promise of training frontier foundation models in a decentralized network of global GPU resources.

Contents

1 Introduction

The rapid scaling of large language models (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2023) has demonstrated unprecedented capabilities but also exposed significant challenges in the infrastructure required for their training. Traditional distributed training approaches rely heavily on high-bandwidth interconnects within centralized data centers. However, there is increasing interest in enabling training across geographically distributed nodes, leveraging standard internet connections to pool GPU resources for collaborative model training. This paradigm shift introduces new challenges: network bandwidth between nodes can be up to three orders of magnitude lower than in typical HPC environments, and the pool of training nodes can be dynamic, with nodes joining or leaving the process unpredictably, making system reliability a critical concern (Ryabinin and Gusev, 2020).

In this paper, we present the first large-scale experiment collaboratively training a 10 billion parameter model over 1T tokens across five countries and three continents on up to 112 H100 GPUs simultaneously. We achieve an overall compute utilization of 83% across continents and 96% when training exclusively on nodes distributed across the entire United States, introducing minimal overhead compared to centralized training approaches. Notably, nodes run independently for approximately 38 minutes before performing an all-reduce operation, which takes around 1 to 7 minutes depending on the configuration, ensuring efficient utilization while minimizing communication overhead. Our results show that INTELLECT-1 can maintain training convergence and high compute utilization despite severe bandwidth constraints and node volatility, opening new possibilities for decentralized, community-driven training of frontier foundation models.

We introduce the PRIME framework specifically designed for training large models across unreliable, bandwidth-constrained nodes. PRIME introduces several key innovations to address these challenges. At its core is the `ElasticDeviceMesh`, a novel abstraction that manages both fault-tolerant communication across the internet and efficient local communication within nodes. This hybrid approach combines the benefits of Fully Sharded Data Parallel (FSDP) (Zhao et al., 2023) training for intra-node efficiency with the Distributed Low-Communication (DiLoCo) (Douillard et al., 2024; Jaghouar et al., 2024) algorithm for minimal inter-node communication.

To maximize training efficiency in this challenging environment, PRIME implements several crucial optimizations. We implement an efficient CPU-offloaded version of DiLoCo that does not add more GPU memory overhead than normal distributed training. Our system employs quantization of gradient transfers to 8-bit integers, reducing communication volume by up to $2000\times$ to standard data parallel training by combining int8 quantization of gradients with an outer optimizer synchronization every 500 steps. Furthermore, PRIME features robust fault tolerance mechanisms, including dynamic node addition and removal, and efficient checkpoint recovery protocols.

Using PRIME, we successfully collaborated to train INTELLECT-1 on a high-quality dataset comprising 1 trillion tokens. This achievement was made possible through the contributions of 30 independent compute sponsors, including industrial partners and individual supporters, who provided H100 nodes from around the globe. The training process was accompanied by a public dashboard[1], allowing real-time monitoring of performance.

We open-source the INTELLECT-1 base model, intermediate checkpoints, pre-training data, post-trained checkpoints, post-train data at `huggingface.co/PrimeIntellect/INTELLECT-1` and the PRIME framework at `github.com/PrimeIntellect-ai/prime`.

The remainder of this report is organized as follows: Section 2 provides a detailed overview of the key features of the PRIME framework. Section 3 describes the experimental setup for INTELLECT-1, compute efficiency analysis, training behavior, post-training techniques, and model evaluation. Section 4 discusses possible implications of decentralized training for the open-source AI ecosystem. Finally, Section 5 concludes the report and outlines directions for future work.

2 Prime Framework: Enabling Scalable Decentralized Training

Modern distributed training of large language models typically relies on high-bandwidth interconnects, with InfiniBand networks providing up to 3.2 Tb/s communication speeds between nodes. However, when training needs to be distributed across geographically distant locations, such high-bandwidth

[1]Public training dashboard: `https://app.primeintellect.ai/intelligence`

connections are unavailable. In our setting, the maximum available bandwidth between nodes is limited to 500 Mb to 4 Gb/s — three orders of magnitude lower than typical HPC environments. This severe bandwidth constraint makes communication the primary bottleneck for distributed training, necessitating new approaches to efficient model synchronization.

To address this constraint, we propose an implementation of the Distributed Low-Communication (DiLoCo) algorithm (Douillard et al., 2024). DiLoCo has been shown to enable training across poorly connected devices while maintaining convergence comparable to traditional distributed training (Jaghouar et al., 2024). For further bandwidth reduction, we experiment with performing 8-bit quantization of pseudo-gradients, which combined with DiLoCo's reduced synchronization frequency can provide up to $2000\times$ reduction in communication volume compared to standard data parallel training while maintaining model quality.

DiLoCo performs a version of Local SGD (Stich, 2019), where each independent worker executes H local optimization steps using AdamW before synchronizing through an outer optimization step using Nesterov momentum. The key insight is that communication between workers is required only every H steps (typically up to 500), dramatically reducing bandwidth requirements compared to traditional data parallel training that communicates gradients at every step.

Algorithm 1 DiLoCo Algorithm

Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`
for outer step $t = 1 \ldots T$ **do**
  for worker $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for inner step $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization
      $\theta_i^{(t)} \leftarrow \texttt{InnerOpt}(\theta_i^{(t)}, \nabla\mathcal{L})$
    end for
  end for
  $\Delta^{(t)} \leftarrow \frac{1}{k}\sum_{i=1}^{k}(\theta^{(t-1)} - \theta_i^{(t)})$ ▷ Averaging outer gradients
  $\theta^{(t)} \leftarrow \texttt{OuterOpt}(\theta^{(t-1)}, \Delta^{(t)})$ ▷ Outer optimization
end for

In this section, we detail our implementation that enables efficient DiLoCo training across bandwidth-constrained environments.

2.1 Efficient DiLoCo Implementation

PRIME offers an efficient implementation of DiLoCo that maintains the same GPU memory footprint as standard distributed training. Pseudo-gradient computation is offloaded to the CPU, followed by a custom IP-based ring all-reduce (Thakur et al., 2005) implementation, which is required for efficient aggregation of pseudo-gradients. While the time taken for the outer all-reduce operation can be up to multiple minutes depending on the specific connectivity of the peers, its infrequent nature still allows for high overall GPU utilization.

The host-offloaded outer optimization process consists of the following steps:

1. Compute pseudo-gradients and perform all-reduce synchronization
2. Execute the outer optimizer step given the reduced pseudo-gradients on CPU
3. Transfer the updated weights back to GPU workers
4. Retain the model weights on CPU for pseudo-gradient delta computation of the next step

This implementation requires additional CPU memory to store the model parameters, outer optimizer states, and pseudo-gradients. While this represents an increased memory burden, modern GPU servers typically have abundant system memory, making this trade-off acceptable.

2.2 Int8 Ring-All-Reduce

To lower the required communication bandwidth of the pseudo-gradient commmunication, we implement a specialized int8 quantized ring-all-reduce (Thakur et al., 2005) kernel with fp32 accumulation.

Communicating quantized int8 values instead of the original fp32 values results in a 4 times reduction in communication payload. We employ the uniform quantization strategy with clipping from Ryabinin et al. (2020):

1. For each tensor, we compute the mean (μ) and standard deviation (σ)
2. The quantization range is then set to $[\mu - 6\sigma, \mu + 6\sigma]$.
3. This range is uniformly divided into 256 buckets.
4. The codebook values are determined by computing the average value within each bucket.

This approach effectively handles outliers while preserving the distribution of values within the most significant range of the tensor.

Naive application of the reduce function in the quantized format will result in numeric precision losses. Specifically, $(Q(a) + Q(b)) \neq Q(a + b)$, where a and b are fp32 values and Q represents the above quantization function. Therefore, we merely quantize the reduce-terms during transmission while performing the reduction operation in full precision (fp32).

This approach requires significant compute overhead for quantization and dequantization. We alleviate this by performing pipelined execution of the ring all-reduce, allowing the quantization and dequantization compute overhead to be overlapped with the communication overhead by scheduling them asynchronously. This hides much of the quantization overhead with the communication time, resulting in a minimal increase in the wall clock time.

In order for our implementation to be fast enough to fully utilize our target maximum bandwidth of 4 Gb/s, we implement custom multithreaded uint8 operations in C++ to perform the quantization and dequantization operations, improving the quantization speed by more than 60 times compared to the PyTorch implementation. These optimizations prevent quantization from becoming a bottleneck while maintaining the benefits of reduced communication volume from int8 transmissions.

Quantization during training is known to degrade model performance at scale, particularly over extended training periods, due to the emergence and accumulation of outliers (Dettmers et al., 2022; Kumar et al., 2024). However, our approach differs fundamentally from traditional weight quantization: instead of quantizing the model weights directly, we quantize the pseudo-gradients, which represent the difference between the model states at different timestamps. We argue that quantizing these temporal differences is inherently more robust than quantizing the weights themselves, as it captures the relative changes in outliers rather than their absolute values. Furthermore, in the DiLoCo framework, communication reduction through quantization can be balanced against increasing the number of local steps, offering flexibility in optimizing the communication-convergence tradeoff.

2.3 Hybrid FSDP and DiLoCo

We implement a hybrid approach combining FSDP (Fully Sharded Data Parallellism) within nodes and DiLoCo across nodes. Each node performs local gather and all-reduce operations via fast intra-node peer-to-peer communication to facilitate FSDP and local data parallelism, while DiLoCo handles the inter-node synchronization over IP. Model parameters and optimizer states remain sharded, with each rank responsible for managing associated system memory state necessary for pseudo-gradient computation and the outer optimizer.

Communication for the all-reduce operation of the outer optimizer is orchestrated such that only ranks responsible for the same shard communicate. Ranks responsible for different shards of data do not communicate via DiLoCo and will receive the updated weights for other regions only via the gather operations associated with FSDP.

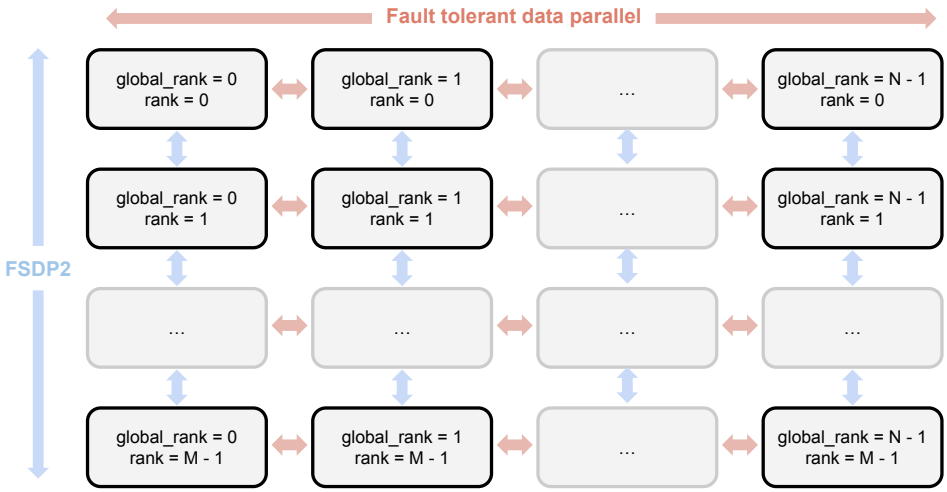

Figure 1: The topology of the `ElasticDeviceMesh`. Each process in the `ElasticDeviceMesh` is assigned a local and global rank. The local rank is used by the FSDP process groups, while the global rank is used by an independent fault-tolerant data-parallel process group.

This hybrid approach allows us to leverage both the memory efficiency of FSDP for local training and the communication efficiency of DiLoCo for cross-node synchronization. Within each node, FSDP can take advantage of high-bandwidth GPU interconnects (NVLink/SXM) for efficient intra-node communication, while DiLoCo minimizes the impact of slower inter-node network speeds.

2.4 Fault Tolerance and Dynamic Node Management

This section describes PRIME's architecture for handling dynamic node participation in distributed training. We focus on two key capabilities: (1) allowing new nodes to join an ongoing training session without disrupting active nodes, and (2) maintaining training continuity when nodes fail or leave the training run.

2.4.1 Dynamic Process Groups

Our implementation focuses specifically on data-parallel training configurations, where the primary collective communication operation is all-reduce.

Data-parallel training is uniquely suited for dynamic world sizes because:

1. The all-reduce operation naturally accommodates varying numbers of participants.
2. Node ranks can be assigned arbitrarily, since the all-reduce operation treats all participants equally.
3. Model state is replicated across all nodes without complex sharding schemes.

These properties make data-parallel training significantly more flexible than other parallelism strategies when handling dynamic node membership.

2.4.2 Peer to Peer Checkpoint Transmission

When a new node joins an ongoing training session, it must synchronize its model and optimizer states with the existing cluster through peer-to-peer checkpoint transfer. LocalSGD variations offer a significant advantage over data-parallel training approaches, which do synchronization at every step, by being able to overlap this synchronization with the computation of local steps. Rather than relying on a centralized storage system, we implement two peer-to-peer synchronization strategies:

1. Non-blocking synchronization: The joining node directly downloads the checkpoint from any available active peer while training continues. Upon completion, the new node skips the current set of inner steps and joins at the next outer step with zero pseudo-gradients. This approach maximizes cluster utilization by avoiding training interruptions.
2. Blocking synchronization: Active nodes pause training while the new peer downloads the checkpoint directly from one of the active nodes. This ensures perfect synchronization across all nodes at the cost of temporary training pause.

While the non-blocking approach theoretically offers better cluster utilization, our experiments showed that the blocking strategy was more practical for our setup. Although non-blocking synchronization worked without causing training instability, we observed small loss spikes in the new joiners' initial steps. Given that we added new nodes relatively infrequently (every few days) and peer-to-peer checkpoint transfers completed within 30-60 minutes, we opted for the more conservative blocking approach to ensure maximum training stability.

2.4.3 Node Removal and Failures

Nodes may leave the training process for two primary reasons. A node may leave gracefully through a planned exit, or it might crash unexpectedly. To handle both scenarios, we implement a heartbeat mechanism, where each running node maintains a subprocess that sends a heartbeat signal to the master key-value store at 2-second intervals. Nodes that fail to send a heartbeat for 6 seconds are automatically evicted from the process group. In the case of a graceful exit, the node sends a "deathrattle" signal to the master key-value store, triggering immediate removal from the process group without waiting for the timeout. For crash scenarios, we implement a best-effort attempt to send a deathrattle before the crash occurs. If this fails, the heartbeat timeout mechanism serves as a fallback, eventually removing the unresponsive node.

2.4.4 Parallel TCP Stores

We implemented multiple parallel TCP stores to manage distributed communication, with each store corresponding to a specific replica group. A replica group consists of nodes sharing the same data-parallel rank, thus containing identical shards of the model, gradient, and optimizer state. This partitioning approach was designed to reduce network congestion by isolating communication within relevant groups rather than broadcasting all updates across a single shared store. However, this architecture introduced synchronization challenges. With separate TCP stores operating independently, maintaining consistent state across all replica groups became more complex. The isolation that provided performance benefits also made it more difficult to ensure all groups remained properly synchronized, particularly during node failures or dynamic group resizing operations.

2.4.5 Retries, Timeouts and Edge Cases

The distributed nature of the system necessitates robust handling of failures during collective operations. When a node fails during an all-reduce operation, we implement a retry mechanism that excludes the failed node and attempts to complete the operation with the remaining healthy nodes. A particularly challenging edge case emerges when node failure affect only a subset of replica groups. In this scenario, processes running on the same physical node may become desynchronized, leading to misaligned timeout windows. This desynchronization can result in undefined behavior as different processes attempt to proceed with different world views of the available nodes. The timing issues are exacerbated by the parallel TCP store architecture, as each store may detect and react to failures independently. This can lead to a cascade of timing misalignments across the system. Current timeout values are set empirically based on observed network latencies and failure detection windows, but maintaining consistent timeout behavior across all components remains an open challenge.

2.5 Networking

All nodes involved in the training run were connected through a shared Virtual Private Network (VPN) established using Tailscale[2], a VPN service based on WireGuard.

[2] `https://tailscale.com/`

We utilize a VPN because our inter-node collective communication library, Gloo[3], requires all participating processes to be reachable via a private network interface. Additionally, the VPN ensures secure, encrypted communication between nodes.

Because our code utilizes independent process groups for each rank, we are able to open multiple connections between nodes. This seems to have provided a noticeable increase in the communication bandwidth in our setup.

One issue we observed in the first few days of the training was that the bandwidth between nodes can be quite unreliable. This means that a high-bandwidth connection might suddenly become slow randomly and for no apparent reason. This would result in the high-bandwidth sequence we pick for the ring-all-reduce to become low-bandwidth. We may also experience this issue when nodes leave the training and change the ring topology.

To optimize the performance of global all-reduce operations, we run a background process that continuously measures the bandwidth between nodes. Using these bandwidth measurements, the process finds the optimal ring-order of nodes such that the minimum bandwidth along the cycle is maximized. This is achieved by solving a variation of the Traveling Salesperson Problem, where the goal is to find a Hamiltonian cycle (a cycle that visits every node exactly once and returns to the starting node) that maximizes the minimum edge bandwidth in the cycle. This can be expressed as:

$$\max_{C \in \mathcal{C}} \min_{(u,v) \in C} w(u, v),$$

where $\mathcal{C}$ is the set of all Hamiltonian cycles in the graph $G = (V, E)$ and $w(u, v)$ is the bandwidth of the edge (u, v). The nodes then use this optimized ring-order for all-reduce operations.

During our testing, we observe varying bandwidths when using Tailscale. Interestingly, there were cases where bandwidth with Tailscale was higher compared to standard internet routing. However, we also encountered instances where bandwidth was noticeably lower when using Tailscale. We hypothesize that these variations are influenced by internet routing dynamics. Tailscale leverages its internal network infrastructure for some connections, which can either outperform or underperform standard internet routing governed by Border Gateway Protocol (BGP). Factors such as latency, route optimization, and network congestion likely contribute to these observed differences.

Ultimately, our analysis revealed that Tailscale became a bottleneck compared to normal IP-based communication, particularly for intercontinental data transfer. We believe that implementing or leveraging an All-Reduce library that operates over public IP addresses, similarly to Hivemind (Ryabinin et al., 2020), would enhance our training performance.

2.6 Future work

The INTELLECT-1 training experiment has demonstrated the need for completely replacing the established collective communications software stack for the realm of internet communication, as intra-datacenter communication is tailored to very different needs that do not translate to global internet communication. Thus, we have started work on our own collective communications library named PCCL ("Prime Collective Communications Library"), which implements a custom TCP-based network protocol called "Collective Communications over IP", which will automatically handle peers joining and leaving gracefully, synchronization of designated shared state and bandwidth-aware topology optimization.

[3] `https://github.com/facebookincubator/gloo`

3 INTELLECT-1 Training

3.1 Experimental Setup

Using PRIME, we conducted a public training run of a 10 billion parameter model distributed across up to eight non-colocated data centers spanning three continents (North America, Europe, and Asia). The model is based on the Llama 3 architecture (Grattafiori et al., 2024). It was pre-trained with a sequence length of $8,192$ on a total of $1T$ tokens with a high quality data mix (see Table 1 for details). We employ a weight-space decay (WSD) learning rate scheduler (Hägele et al., 2024) with an inner learning rate of $7.5e^{-5}$ for the AdamW optimizer. The WSD learning rate scheduler maintains a constant learning rate after an initial warm-up phase and offered us flexibility in the number of tokens we can train on, depending on the number of compute contributions. During the final 20% of training, we applied a learning rate cool-down phase along with additional post-training optimizations. For optimal performance, we chose a conservative number of 100 inner steps between each DiLoCo synchronization, which, on clusters of 8xH100 nodes, took roughly 38 minutes to complete. We quantize the pseudo-gradients to int8, reducing communication requirements by a factor of $400\times$. The average all-reduce synchronization for the DiLoCo outer optimizer using PRIME took approximately 2 minutes for nodes across the United States and 10 minutes for global training, limiting communication between nodes to just 2-10% of the total training time. The model was trained using an auxiliary max-z-loss (Yang et al., 2023) for improved training stability. For more details about the model configuration, see Appendix A.

This experimental setup represents the first successful training of a model of this scale across such geographically distributed compute resources with limited network bandwidth.

In the following subsections, we analyze the compute efficiency across different geographical distributions, examine the system's resilience to node changes, detail the pre-training process and results, describe our post-training optimization techniques, and present comprehensive evaluation results against comparable models.

3.2 Compute Efficiency Analysis

We analyze the compute efficiency across three geographical distributions of training nodes: within the United States (USA), between USA and Europe (Transatlantic), and across USA, Europe, and Asia (Global). Table 2 presents the Model FLOPS Utilization (MFU) (Chowdhery et al., 2022) and timing breakdown for each scenario.

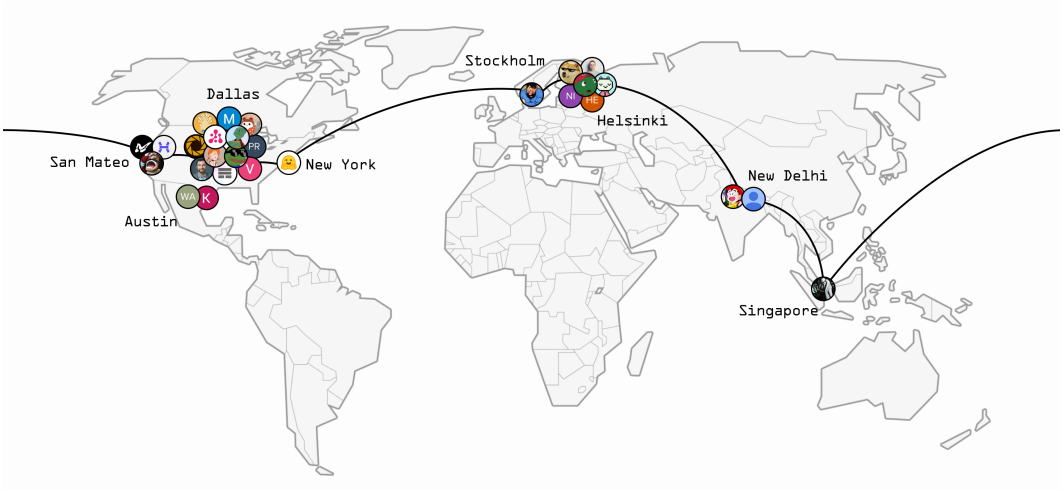

Figure 2: Locations of the nodes by all 30 compute contributors for INTELLECT-1. The lines between nodes illustrate the Ring-All-Reduce topology, spanning the whole globe from the US to Europe, Asia, and back to the US.

Dataset Name	Weight (%)	Weight For Annealing Phase (%)
FineWeb-Edu (Penedo et al., 2024)	55	80
FineWeb (Penedo et al., 2024)	10	10
StackV1-popular (Kocetkov et al., 2022)	20	10
DCLM-baseline (Li et al., 2024)	10	0
OpenWebMath (Paster et al., 2023)	5	0

Table 1: **Pre-training Dataset Composition:** The model was trained on a diverse mixture of high-quality datasets weighted as follows. As some of these datasets contain shards that contain correlated data, we pre-shuffled the datasets by random sampling from 12 streaming dataset iterators and resharding the dataset. The total number of tokens in our data mix, processed with the Llama-3 tokenizer, consists of over 6 trillion tokens. We adjusted the data mix to prioritize dataset segments with substantially higher loss for the learning rate annealing phase of the training.

Scenario	MFU (%)	Inner step time, min	Median All-Reduce time, s	Compute Util (%)
Baseline (no comm)	43.3	38	-	100
USA	41.4	38	103	95.7
USA + Europe	37.1	38	382	85.6
Global	36.0	38	469	83.0

Table 2: Performance metrics for training across different geographical configurations. Compute utilization refers to the proportion of time the training is not communicating with other nodes.

The baseline training configuration without communication achieves 43% MFU, which decreases to 41.4% for USA-only distribution. When extending to transatlantic training, MFU drops to 37.1%, and further decreases to 36.2% in a global setting. The computation time for 100 inner steps remains constant at 38 minutes across all scenarios. However, the all-reduce communication time varies significantly: the median is 103 seconds within the USA, increasing to 382 seconds for transatlantic communication, and reaching 469 seconds in the global setting. For reference, checkpoint saving to disk takes 60 seconds. The overhead from CPU-based pseudo-gradient computation and outer optimizer steps is negligible at 5-10 seconds

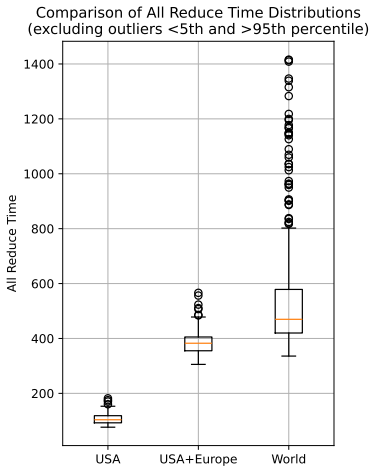

Figure 3: Distribution of all-reduce operation times across different geographical configurations. The variance increases significantly as we move from USA-only to global distribution, indicating less reliable network conditions.

Beyond the absolute performance implications of different data types and buffering strategies, our experiments revealed significant variability in network reliability across geographical distributions. As shown in Figure 3, the variance in all-reduce completion times increases dramatically as we

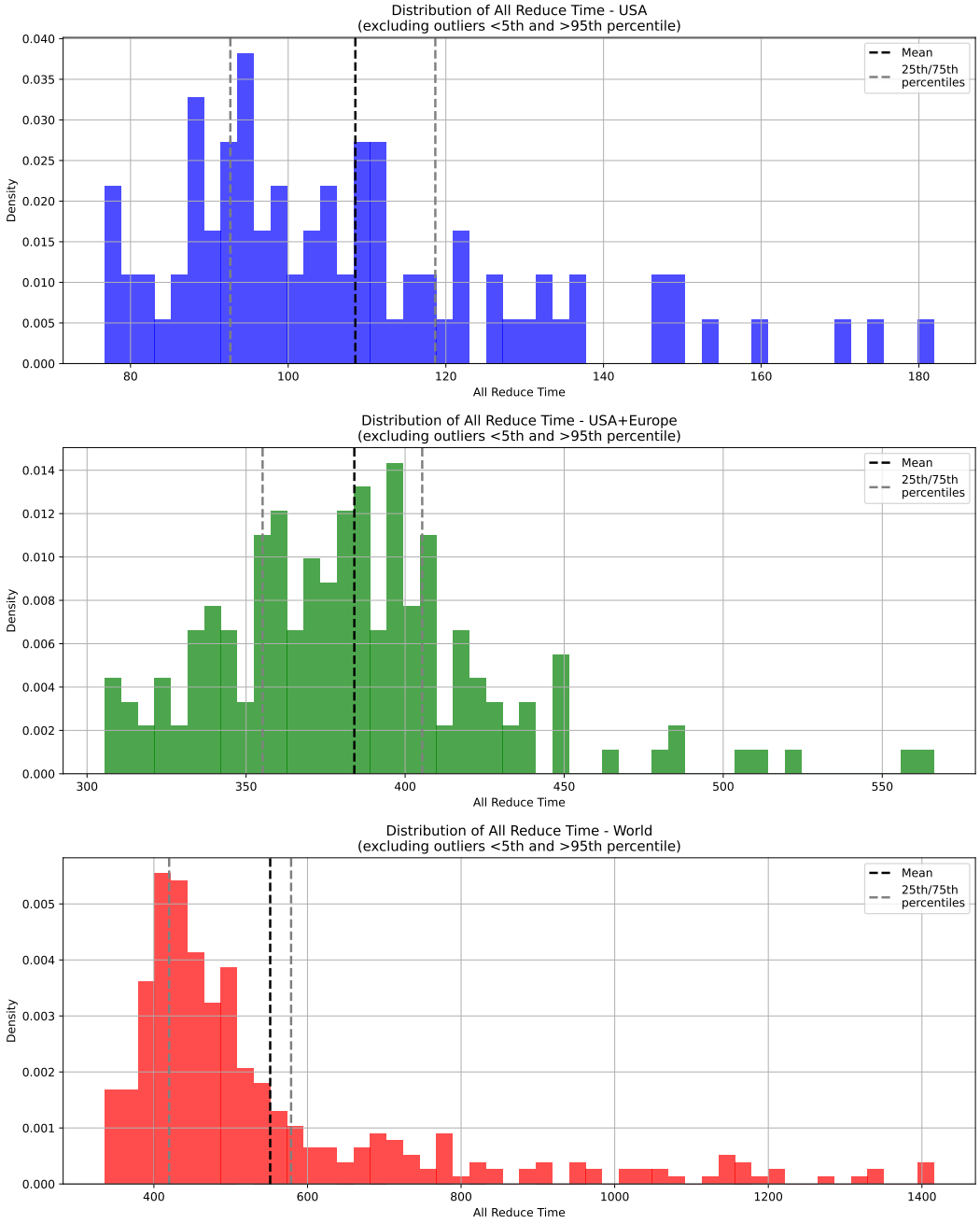

Figure 4: Distribution of all-reduce completion times across different geographical setups. The increasing spread and right-skewed nature of the distributions highlight growing network instability as geographical distances increase. Red represents global, green represents USA and Europe, and blue represents USA-only training.

expand from USA-only to global distribution. While USA-only operations show relatively tight bounds, global operations exhibit extensive outliers, with some operations taking up to 2x longer than the median time. Figure 4 further illustrates this pattern through the probability distributions of completion times, showing increasingly heavy right tails as geographical distance grows. This network instability underscores the importance of robust buffering and quantization strategies to minimize the impact of network variability on training performance.

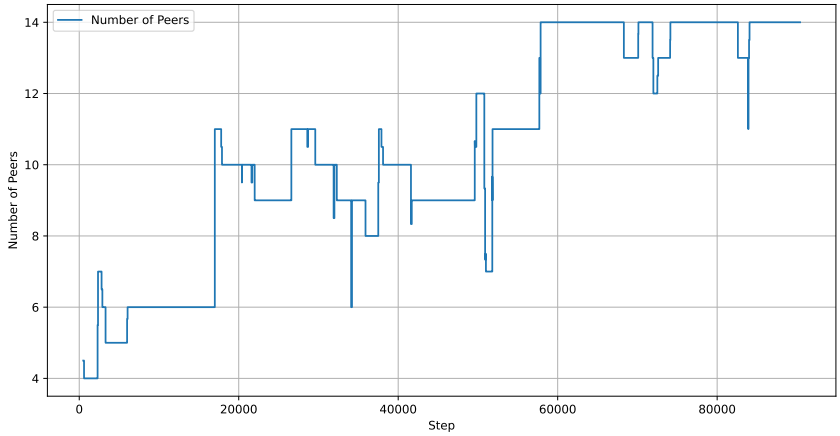

Figure 5: Number of active training nodes over training steps. The graph demonstrates PRIME's ability to handle dynamic node participation, starting with 4 nodes and scaling up to 14 nodes, while maintaining training stability despite frequent node fluctuations.

3.3 Analysis of Resilience to Node Changes

Our fault-tolerant implementation of DiLoCo demonstrates robust fault tolerance capabilities in distributed training environments. As shown in Figure 5, our training process began with just 4 nodes and gradually scaled up to 14 nodes over time. The system successfully maintained training stability while handling both the gradual onboarding of new nodes and occasional node departure and failures, with the number of active peers fluctuating throughout the training process.

While DiLoCo demonstrates robust adaptation to gradual node changes, we observed that simultaneous loss of multiple nodes (in our case, 4 out of 12) can impact training stability. In this specific case, we opted to resume from a checkpoint. Although such large-scale simultaneous node failures are rare in practice, future work could explore additional mechanisms to handle extreme scenarios where a significant portion of the training fleet is lost at once.

3.4 Pre-training

The pre-training of INTELLECT-1 for 1 trillion tokens took place over 42 days, from October 10, 2024, to November 22, 2024. dFigure 6 presents the loss curve over the training steps, demonstrating stable convergence despite significant dynamic on-/off-boarding of compute resources. Around 80% of the way through training (step $74,700$), the learning rate annealing phase began. At this stage, we adjusted the data mix to prioritize dataset segments with substantially higher loss, thereby reducing the weight of the Stack (Kocetkov et al., 2022) and OpenWebMath (Paster et al., 2023) in the mix. This adjustment accounts for the increased loss observed at that point, as the loss on code was significantly lower than text.

3.5 Post-training

After completing the globally distributed pretraining phase, we applied several post-training techniques to enhance INTELLECT-1's capabilities and task-specific performance. Our post-training methodology consisted of three main phases.

First, we conducted an extensive series of 16 Supervised Fine-Tuning (SFT) trainings, with individual runs ranging from 1 to 3.3 billion tokens each. The most successful configuration used 2.4 billion training tokens over 3 epochs. We used `MergeKit`, `EvolKit`, and `DistillKit` from Arcee AI to combine the models, generate the data sets, and distill the logits, respectively. For training data, we used a diverse set of high-quality datasets:

1. **New Datasets** (released with INTELLECT-1):
    - `arcee-ai/EvolKit-75k` (generated via `EvolKit`)

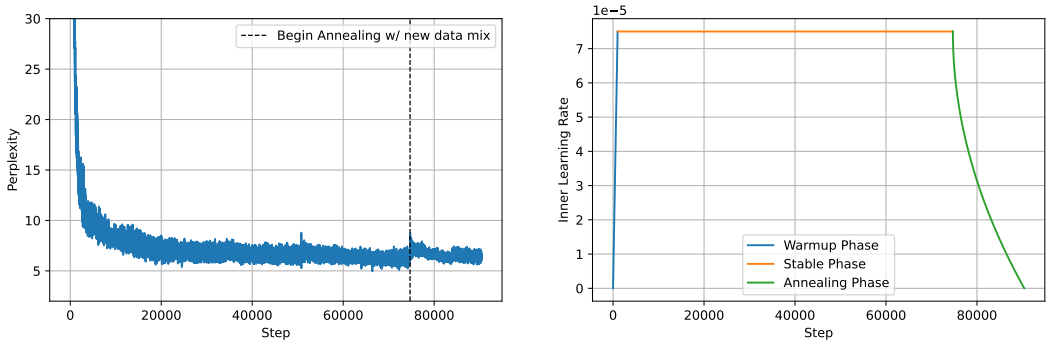

Figure 6: Training dynamics showing model perplexity and learning rate over training steps, including warmup, stable, and annealing phases. The left graph demonstrates the decrease in perplexity from initial training through convergence, while the right graph illustrates the learning rate schedule with distinct warmup, stable, and annealing phases.

- `arcee-ai/Llama-405B-Logits`
- `arcee-ai/The-Tomb`

2. **Instruction Following**:
   - `mlabonne/open-perfectblend-fixed` (generalist capabilities)
   - `microsoft/orca-agentinstruct-1M-v1-cleaned` (Chain-of-Thought)
   - `Post-training-Data-Flywheel/AutoIF-instruct-61k-with-funcs`
3. **Domain-Specific**:
   - `Team-ACE/ToolACE` (function calling)
   - Synthia coder (programming)
   - `ServiceNow-AI/M2Lingual` (multilingual)
   - `AI-MO/NuminaMath-TIR` (mathematics)
4. **Tulu-3 Persona Datasets**:
   - `allenai/tulu-3-sft-personas-code`
   - `allenai/tulu-3-sft-personas-math`
   - `allenai/tulu-3-sft-personas-math-grade`
   - `allenai/tulu-3-sft-personas-algebra`

Second, we execute 8 distinct Direct Preference Optimization (DPO) runs with various combinations of data sets to enhance specific performance metrics and align the model with human preferences. A key advantage in our post-training process was INTELLECT-1's use of the Llama-3 tokenizer, which allowed us to utilize logits from Llama-3.1-405B to heal and maintain precision during the post-training process via `DistillKit`.

Finally, we performed 16 strategic merges between candidate models using `MergeKit` to create superior combined models that leverage the strengths of different training runs. During the post-training phase, we observed that when using a ChatML template without an explicit BOS (begin-of-sequence) token, the initial loss was approximately 15. However, when switching to the Llama 3.1 chat template, the loss for these trainings started much lower at approximately 1.1, indicating better alignment with the underlying Llama 3 tokenizer.

The combination of these post-training techniques resulted in significant improvements in various benchmarks (detailed in Table 4), particularly in knowledge retrieval, grade school math, instruction following and reasoning.

3.6 Evaluation

This section presents the main benchmark results for INTELLECT-1. We compare with similarly sized open-source models which were pre-trained in a centralized setting on similar amounts of tokens.

For the base model evaluation, we use eight benchmarks commonly featured in evaluations, such as the Hugging Face Open LLM Leaderboard. These include MMLU 5-shot (Hendrycks et al., 2021), HellaSwag 10-shot (Zellers et al., 2019), ARC-Challenge 25-shot (Clark et al., 2018), GPQA (Rein et al., 2023), GSM8K 5-shot (Cobbe et al., 2021), TruthfulQA (Lin et al., 2022), WinoGrande 5-shot (Sakaguchi et al., 2019) and Big Bench Hard (BBH) (Suzgun et al., 2022). The results are shown in Table 3. Additionally, for the post-trained models, we run additional benchmarks from the Hugging Face Open LLM Leaderboard 2 such as IFEval (Zhou et al., 2023), and compare to other post-trained and instruction-tuned models in table 4. All evaluations are performed with the Language Model Evaluation Harness framework (Gao et al., 2024). The benchmark results are promising for a 10B model trained on a trillion tokens, providing strong evidence that our DiLoCo approach effectively scales to the 10B model size.

Model	**Size**	**Tokens**	**MMLU**	**HellaSwag**	**ARC-C**
INTELLECT-1	10B	1T	37.5	72.26	52.13
MPT-7B (Team, 2023)	7B	1T	26.8	77.41	46.67
Falcon-7B (Almazrouei et al., 2023)	7B	1.5T	26.2	78.23	47.61
Pythia-12B (Biderman et al., 2023)	12B	300B	26.5	68.83	40.61
LLM360-Amber (Liu et al., 2023)	7B	1.3T	24.5	74.08	42.75
LLaMA-7B (Touvron et al., 2023a)	7B	1T	35.1	78.19	50.43
LLaMA-13B (Touvron et al., 2023a)	13B	1T	46.9	81.05	56.14
LLaMA2-7B (Touvron et al., 2023b)	7B	2T	45.3	78.64	54.10
LLaMA2-13B (Touvron et al., 2023b)	13B	2T	54.8	82.58	59.81
	GPQA	**GSM8K**	**TruthfulQA**	**Winogrande**	**BBH**
INTELLECT-1	26.12	8.1	35.47	65.82	32.97
MPT-7B	25.67	8.3	33.43	71.11	32.88
Falcon-7B	23.66	4.9	34.28	70.32	33.00
Pythia-12B	24.33	4.09	31.83	65.27	31.66
LLM360-Amber	27.01	4.32	40.80	65.35	31.95
LLaMA-7B	23.21	9.7	34.33	72.06	32.86
LLaMA-13B	26.34	17.3	39.48	76.16	39.74
LLaMA2-7B	25.89	13.5	38.75	74.03	34.46
LLaMA2-13B	25.67	24.3	37.38	77.35	41.68

Table 3: Base model evaluation results across various benchmarks, measured against similarly sized open-source models pre-trained in a centralized setting on comparable amounts of total tokens.

Model	**Size**	**Tokens**	**MMLU**	**HellaSwag**	**ARC-C**
INTELLECT-1-INSTRUCT	10B	1T	49.89	71.42	54.52
MPT-7B-Chat	7B	1T	36.29	75.88	51.02
Falcon-7B-instruct	7B	1.5T	25.21	70.61	45.82
LLM360 AmberChat	7B	1.4T	36.02	73.94	43.94
LLaMA2-7B-chat	7B	2T	47.20	78.69	53.33
LLaMA2-13B-chat	13B	2T	53.51	82.47	59.73
	GPQA	**GSM8K**	**TruthfulQA**	**BBH**	**IFEval**
INTELLECT-1-INSTRUCT	28.32	38.58	42.16	34.85	40.39
MPT-7B-Chat	26.79	8.26	35.22	32.30	14.39
Falcon-7B-instruct	26.34	4.93	44.13	31.98	24.82
LLM360 AmberChat	27.23	6.14	40.80	31.14	18.71
LLaMA2-7B-chat	28.57	23.96	45.58	35.50	45.80
LLaMA2-13B-chat	28.35	37.15	44.12	39.05	46.88

Table 4: Post-trained model evaluation results across various benchmarks.

While INTELLECT-1 demonstrates encouraging benchmark results and represents the first model of its size successfully trained on a decentralized network of GPUs, it still lags behind current state-of-the-art models trained on an order of magnitude more tokens. Future work will focus on scaling the model series with significantly larger compute budgets, number of contributors, introducing novel architectural advancements beyond Llama 3, and leveraging higher-quality datasets.

4 Discussion: Decentralized Training

The successful training of INTELLECT-1 demonstrates a key technical advancement in enabling a more open and decentralized AI ecosystem, with significant implications for the future development and governance of advanced artificial intelligence systems.

The current trajectory of AI development shows increasing consolidation of computational resources and research capabilities to a small number of corporate and state entities. As the cost of training competitive frontier models increases, this consolidation only accelerates. While centralization has the benefit of reduced coordination costs, and increased efficiency, it has the negative externality of creating significant points of control and increased risk of capability capture or misuse.

Open-source AI currently serves as the best counterweight to a closed-source AI ecosystem, and has yielded significant advancements in virtually every technical aspect of artificial intelligence development. However, open-source AI has yet to amass the scale of compute that is relevant to achieve parity with the largest closed-source AI labs. This is a requirement if the open-source ecosystem intends to train and open-source competitive frontier models. As demonstrated by this work and prior results on collaborative deep learning (Pascutto and Linscott, 2019; Diskin et al., 2021; Borzunov et al., 2022), decentralized training has the potential to enable that scale of compute by making it possible to pool resources across the world for training large-scale foundational models.

Historical precedent from distributed compute protocols and incentive networks demonstrates the potential scale of community-pooled compute resources. Bitcoin's network grew from a few personal computers to over 10 Gigawatts of compute in 12 years, and Ethereum's network reached similar scale before transitioning to proof of stake. These networks demonstrate how properly aligned economic incentives can mobilize massive computational resources across geographical and institutional boundaries. Similar dynamics could emerge in decentralized AI training: with appropriate incentive mechanisms and technical infrastructure, community-pooled AI training could potentially exceed the compute capacity of any centralized training facilities.

5 Conclusion

In this report, we introduce INTELLECT-1, the first globally trained language model at the 10-billion-parameter scale. We are open-sourcing the base model, intermediate checkpoints, pre-training data, post-training checkpoints, and post-training data.

Alongside the INTELLECT-1 release, we are also open-sourcing PRIME, a scalable distributed training framework designed for fault-tolerant, high-performance training on unreliable, globally distributed nodes with low network bandwidth.

We hope that this report, along with our open-source releases, will contribute to the broader ecosystem, advancing efforts to make globally distributed training a viable option for developing state-of-the-art open models.

Acknowledgements

We would like to thank all the companies and individuals who generously provided compute resources for the training run from locations all around the world. This includes Arcee AI, @oldmankotaro, @skre_0, @marloXBT, @rodeo_crypro, @herb0x_, Autonolas, @Etherean007, Hugging Face, @mev_pete, 0xfr_, @iroh_pm, kiteman, realtek, primeprimeint1234, Hyperbolic, NWO, hecataeus, VirtualMachine, @notdroll, SemiAnalysis, waiting__, @toptickcrypto, sto, washout_segment_0b and klee.

We would also like to thank Arthur Douillard for their work on DiLoCo. Tristan Rice and Junjie Wang for discussions and ideas on fault-tolerant training. Chien-Chin Huang and Iris Zhang for ideas and discussions related to asynchronous distributed checkpointing. Yifu Wang for discussions about Tensor Parallel. Andrew Gu for convincing us to switch to FSDP2 to get rid of some memory allocation issues we were facing with FSDP1.

References

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of open language models, 2023. URL https://arxiv.org/abs/2311.16867.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

Alexander Borzunov, Max Ryabinin, Tim Dettmers, Quentin Lhoest, Lucile Saulnier, Michael Diskin, Yacine Jernite, and Thomas Wolf. Training transformers together, 2022.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, et al. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Quentin Lhoest, Anton Sinitsin, Dmitry Popov, Dmitriy Pyrkin, Maxim Kashirin, Alexander Borzunov, Albert Villanova del Moral, Denis Mazur, Ilia Kobelev, Yacine Jernite, Thomas Wolf, and Gennady Pekhimenko. Distributed deep learning in open collaborations. In *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=FYHktcK-7v.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models, 2024. URL https://arxiv.org/abs/2311.08105.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations, 2024. URL https://arxiv.org/abs/2405.18392.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. Opendiloco: An open-source framework for globally distributed low-communication training, 2024. URL https://arxiv.org/abs/2407.07852.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code, 2022. URL https://arxiv.org/abs/2211.15533.

Tanishq Kumar, Zachary Ankner, Benjamin F. Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision, 2024. URL https://arxiv.org/abs/2411.04330.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, et al. Datacomp-lm: In search of the next generation of training sets for language models, 2024. URL https://arxiv.org/abs/2406.11794.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022. URL https://arxiv.org/abs/2109.07958.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, Richard Fan, Yi Gu, Victor Miller, Yonghao Zhuang, Guowei He, Haonan Li, Fajri Koto, Liping Tang, Nikhil Ranjan, Zhiqiang Shen, Xuguang Ren, Roberto Iriondo, Cun Mu, Zhiting Hu, Mark Schulze, Preslav Nakov, Tim Baldwin, and Eric P. Xing. Llm360: Towards fully transparent open-source llms, 2023. URL https://arxiv.org/abs/2312.06550.

OpenAI. Gpt-4 technical report, 2023.

Gian-Carlo Pascutto and Gary Linscott. Leela chess zero, 2019. URL http://lczero.org/.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023. URL https://arxiv.org/abs/2310.06786.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/file/25ddc0f8c9d3e22e03d3076f98d83cb2-Paper.pdf.

Max Ryabinin, Alexander Borzunov, Michael Diskin, Anton Gusev, Denis Mazur, Vsevolod Plokhotnyuk, Alexey Bukhtiyarov, Pavel Samygin, Anton Sinitsin, and Artem Chumachenko. Hivemind: Decentralized Deep Learning in PyTorch, April 2020. URL https://github.com/learning-at-home/hivemind.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1g2JnRcFX.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

MosaicML NLP Team. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL www.mosaicml.com/blog/mpt-7b. Accessed: 2023-05-05.

Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19: 49 – 66, 2005. URL https://api.semanticscholar.org/CorpusID:90404.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, et al. Llama 2: Open foundation and fine-tuned chat models, 2023b. URL https://arxiv.org/abs/2307.09288.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, JunTao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. Baichuan 2: Open large-scale language models, 2023. URL https://arxiv.org/abs/2309.10305.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A Model Configuration

Table 5: Model Configuration for INTELLECT-1. The model is based on the Llama3 architecture (Grattafiori et al., 2024), with the only difference from Llama3-8B being the number of layers.

PARAMETER	10B
NUMBER OF LAYERS	42
HIDDEN DIM	$4,096$
NUMBER OF HEADS	32
K/V SIZE	8
VOCAB SIZE	$128,256$
INNER LEARNING RATE	$7.5e-5$
WARMUP STEPS	$1,000$
WEIGHT DECAY	0.1
BATCH SIZE	128
SEQUENCE LENGTH	$8,192$
OUTER NESTEROV LR	0.7
OUTER NESTEROV MOMENTUM	0.9
MAX Z LOSS WEIGHT	$2e-4$
ADAM BETA 1	0.9
ADAM BETA 2	0.95
WEIGHT DECAY	0.1