arXiv:2407.21075v1 [cs.AI] 29 Jul 2024

Apple Intelligence Foundation Language Models

Apple

We present foundation language models developed to power Apple Intelligence features, including a ~3 billion parameter model designed to run efficiently on devices and a large server-based language model designed for Private Cloud Compute [Apple, 2024b]. These models are designed to perform a wide range of tasks efficiently, accurately, and responsibly. This report describes the model architecture, the data used to train the model, the training process, how the models are optimized for inference, and the evaluation results. We highlight our focus on Responsible AI and how the principles are applied throughout the model development.

1 Introduction

At the 2024 Worldwide Developers Conference, we introduced Apple Intelligence, a personal intelligence system integrated deeply into iOS 18, iPadOS 18, and macOS Sequoia.

Apple Intelligence consists of multiple highly-capable generative models that are fast, efficient, specialized for our users' everyday tasks, and can adapt on the fly for their current activity. The foundation models built into Apple Intelligence have been fine-tuned for user experiences such as writing and refining text, prioritizing and summarizing notifications, creating playful images for conversations with family and friends, and taking in-app actions to simplify interactions across apps.

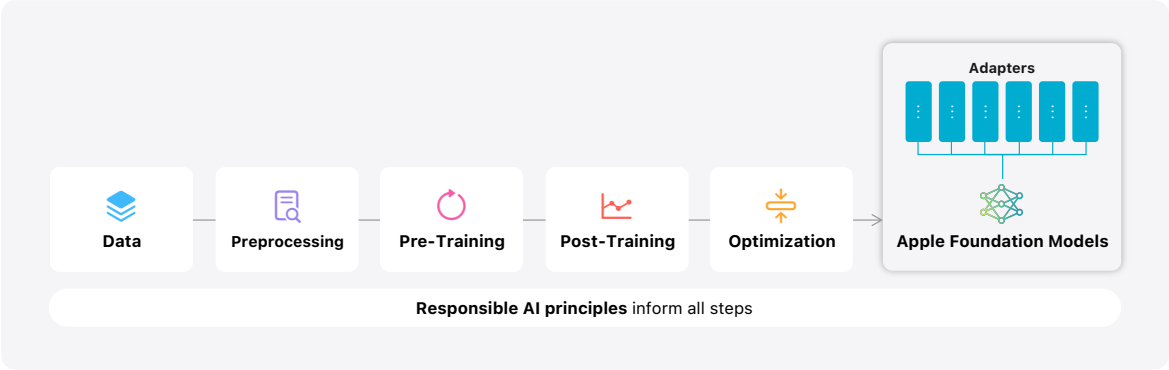

Figure 1: Modeling overview for the Apple foundation models.

In this report we will detail how two of these models—AFM-on-device (AFM stands for *Apple Foundation Model*), a ~3 billion parameter language

model, and AFM-server, a larger server-based language model—have been built and adapted to perform specialized tasks efficiently, accurately, and responsibly (Figure 1). These two foundation models are part of a larger family of generative models created by Apple to support users and developers; this includes a coding model (based on an AFM language model) to build intelligence into Xcode, as well as a diffusion model to help users express themselves visually, for example, in the Messages app.

Apple Intelligence is designed with Apple's core values at every step and built on a foundation of industry-lead privacy protection. Additionally, we have created Responsible AI principles to guide how we develop AI tools, as well as the models that underpin them:

1. **Empower users with intelligent tools**: We identify areas where AI can be used responsibly to create tools for addressing specific user needs. We respect how our users choose to use these tools to accomplish their goals.

2. **Represent our users**: We build deeply personal products with the goal of representing users around the globe authentically. We work continuously to avoid perpetuating stereotypes and systemic biases across our AI tools and models.

3. **Design with care**: We take precautions at every stage of our process, including design, model training, feature development, and quality evaluation to identify how our AI tools may be misused or lead to potential harm. We will continuously and proactively improve our AI tools with the help of user feedback.

4. **Protect privacy**: We protect our users' privacy with powerful on-device processing and groundbreaking infrastructure like Private Cloud Compute. We do not use our users' private personal data or user interactions when training our foundation models.

These principles are reflected at every stage of the architecture that enables Apple Intelligence and connects features and tools with specialized models.

In the remainder of this report, we provide details on decisions such as: how we develop models that are highly capable, fast, and power-efficient; how we approach training these models; how our adapters are fine-tuned for specific user needs; and how we evaluate model performance for both helpfulness and unintended harm.

2 Architecture

The AFM base models are dense decoder-only models that build on the Transformer architecture [Vaswani et al., 2017], with the following design choices:

- A shared input/output embedding matrix [Press and Wolf, 2016] to reduce memory usage for parameters.

- Pre-Normalization [Nguyen and Salazar, 2019] with RMSNorm [Zhang and Sennrich, 2019] for training stability.
- Query/key normalization [Wortsman et al., 2023] to improve training stability.
- Grouped-query attention (GQA) [Ainslie et al., 2023] with 8 key-value heads to reduce the KV-cache memory footprint.
- The SwiGLU activation [Shazeer, 2020] for higher efficiency.
- RoPE [Su et al., 2024] positional embeddings with the base frequency set to 500k for long-context support.

Table 1 provides some details about AFM-on-device dimensions.

Model dimension	3072
Head dimension	128
Num query heads	24
Num key/value heads	8
Num layers	26
Num non-embedding params (B)	2.58
Num embedding params (B)	0.15

Table 1: AFM-on-device dimensions.

3 Pre-training

Our AFM pre-training process plays a critical role in developing highly capable language models to power a host of Apple Intelligence features that can help and support users. We focus on efficiency and data quality at every step in order to pre-train for a high-quality end-to-end user experience with efficient and low-latency models.

3.1 Data

The AFM pre-training dataset consists of a diverse and high quality data mixture. This includes data we have licensed from publishers, curated publicly-available or open-sourced datasets, and publicly available information crawled by our web-crawler, Applebot [Apple, 2024a]. We respect the right of webpages to opt out of being crawled by Applebot, using standard robots.txt directives.

Given our focus on protecting user privacy, we note that no private Apple user data is included in the data mixture. Additionally, extensive efforts have been made to exclude profanity, unsafe material, and personally identifiable information from publicly available data (see Section 7 for more details). Rigorous decontamination is also performed against many common evaluation benchmarks.

We find that data quality, much more so than quantity, is the key determining factor of downstream model performance. In the following, we provide more details about key components of the data mixture.

3.1.1 Web pages

We crawl publicly available information using our web crawler, Applebot [Apple, 2024a], and respect the rights of web publishers to opt out of Applebot using standard robots.txt directives. Plus, we take steps to exclude pages containing profanity and apply filters to remove certain categories of personally identifiable information (PII). The remaining documents are then processed by a pipeline which performs quality filtering and plain-text extraction, more specifically:

1. Body extraction is performed using a combination of Safari's reader mode and the Boilerpipe [Kohlschütter et al., 2010] algorithm.
2. Safety and profanity filtering, using heuristics and model-based classifiers.
3. Global fuzzy de-duplication using locality-sensitive n-gram hashing.
4. Extensive quality filtering using heuristics and model-based classifiers [Kong et al., 2024; Li et al., 2024a].
5. Decontamination against 811 common pre-training benchmarks, filtering entire documents upon 4-13 gram collisions with any of the benchmark datasets, unless the collision-count for a given n-gram reaches a "common-usage" threshold of 1000.

i.e. if they are simply commonly used phrases.

3.1.2 Licensed datasets

We go to lengths to identify and license a limited amount of high-quality data from publishers. These licensed datasets provide a natural source of diverse and high quality long-context data, so we include them as part of the data mixture for the continued and context-lengthening stages of pre-training (see Section 3.2.2 and 3.2.3 for more details). We decontaminate sections of publisher licensed data the same way we decontaminate web pages (Section 3.1.1).

3.1.3 Code

Code data is obtained from license-filtered[1] open source repositories on GitHub. The bulk of the code data covers 14 common programming languages, including: Swift, Python, C, Objective-C, C++, JavaScript, Java, and Go. The data is de-duplicated, further filtered for PII and quality, and decontaminated in the same fashion as in Section 3.1.1.

[1] Using MIT, Apache, BSD, CC0, CC-BY, Unlicensed, ISC, and Artistic Licenses.

3.1.4 Math

We integrate two categories of high-quality data sourced from the web. The first category is a Math Q&A dataset, comprising 3 billion tokens from 20 web domains rich in math content. We extract the questions and answers by identifying relevant tags from HTML pages. The second category is a collection of 14 billion tokens from web pages such as math forums, blogs, tutorials, and seminars. To filter these web pages, we used a specialized pipeline that includes a math tag filter with a collection of 40 strings to identify mathematical templates, a math symbol filter with a collection of 350 Unicode and LaTeX symbols to identify math content, a quality filter powered by a language model classifier specifically designed for math [Kong et al., 2024], and a domain filter that processes all web pages from domains manually labeled by humans. We applied these filters, followed by deduplication, decontamination, and PII removal to produce the final dataset.

3.1.5 Public datasets

We evaluated and selected a number of high-quality publicly-available datasets with licenses that permit use for training language models. Then, we filtered the datasets to remove personally identifiable information before including them in the pre-training mixture.

3.1.6 Tokenizer

We use a *byte-pair encoding* (BPE) tokenizer, following the implementation from SentencePiece. All numbers are split into individual digits and we use byte-fallback to decompose unknown UTF-8 characters into byte tokens. We do not enable Unicode normalization. The total vocabulary size is 100k and 49k tokens for AFM-server and AFM-on-device, respectively.

3.2 Recipe

We break AFM pre-training into three distinct stages: 1. *core* which consumes most of the compute budget, 2. *continued*, where we down-weight the lower-quality bulk web-crawl data, favoring a higher code and math weight instead combined with inclusion of the licensed data described in Section 3.1.2, 3. *context-lengthening* which is similar to another *continued* pre-training stage, but conducted at longer sequence length and with synthetic long-context data included in the mixture.

Details about model quality after each of the three pre-training stages (alongside additional metrics for AFM derived from our internal benchmark implementations) are in Appendix C, and Appendix D examines AFM-server's long-context capabilities.

All three stages use decoupled weight decay [Loshchilov and Hutter, 2019] for regularization, as well as a simplified version of μParam [Yang et al., 2022], similar to what is described as μParam (simple) in [Wortsman et al., 2023]. Thus far we have not found more sophisticated parameter norm controls to be

necessary at these scales. All stages maintain sharded model and optimizer states in `float32`, casting to `bfloat16` for the forward and backward passes for efficiency.

3.2.1 Core pre-training

AFM-server core training is conducted from scratch, while AFM-on-device is distilled and pruned from a larger model.

AFM-server: We train AFM-server from scratch for 6.3T tokens on 8192 TPUv4 chips, using a sequence length of 4096 and a batch-size of 4096 sequences. The batch size was determined using a scaling law fit to model size and compute budget, however we find that downstream results are relatively insensitive to a fairly wide range of batch sizes, and expect that any value between 0.5× and 2× the predicted batch size would have yielded similar results (the predicted optimum was in fact ∼3072, but 4096 allowed for better chip utilization). We perform a learning rate sweep using a proxy model with a model dimension of 768, finding that the optimum learning rate spans 0.01-0.02, so we choose 0.01 to be conservative. Linear layers will have an effective learning rate scaled by ∼0.1 due to the use of μParam (simple).[2]

We use a tuned decoupled weight decay of 3.16e−4, finding that it works well across all tested model sizes and compute budgets. The learning rate schedule includes a linear warm-up for 5000 steps, followed by cosine decay to 0.005 of the peak over the remainder of training. For further details on the optimizer, see Section 3.2.4. Appendix A compares the AFM core pre-training recipe to a more typical configuration.

AFM-on-device: For the on-device model, we found that knowledge distillation [Hinton et al., 2015] and structural pruning are effective ways to improve model performance and training efficiency. These two methods are complementary to each other and work in different ways. More specifically, before training AFM-on-device, we initialize it from a pruned 6.4B model (trained from scratch using the same recipe as AFM-server), using pruning masks that are learned through a method similar to what is described in [Wang et al., 2020; Xia et al., 2023]. The key differences are: (1) we only prune the hidden dimension in the feed-forward layers; (2) we use Soft-Top-K masking [Lei et al., 2023] instead of HardConcrete masking [Louizos et al., 2018]; (3) we employ the same pre-training data mixture as the core phase to learn the mask, training for 188B tokens. Then, during the core pre-training of AFM-on-device, a distillation loss is used by replacing the target labels with a convex combination of the true labels and the teacher model's top-1 predictions, (with 0.9 weight assigned to the teacher's labels), training for a full 6.3T tokens. We observe that initializing from a pruned model improves both data efficiency and the

[2] In scaling law experiments we find that μParam (simple) stabilizes the optimal learning rate as model size increases, although extrapolating to very significantly deeper and/or larger models does exhibit a slight left-shift beyond what is accounted for.

Distillation → Improves MMLU and GSM8K by 5% and 3%

Init from Pruned model → Improves data efficiency and final benchmark results by 0-2%.

final benchmark results by 0-2%, whilst adding distillation boosts MMLU and GSM8K by about 5% and 3% respectively. More detailed ablation results can be found in Appendix B. All training hyper-parameters except for batch-size are kept the same as AFM-server.

3.2.2 Continued pre-training

For both models we perform continued pre-training at a sequence length of 8192, with another 1T tokens from a mixture that upweights math and code, and down-weights the bulk web-crawl. We also include the licensed data described in Section 3.1.2. We use a peak learning rate of 3e−4 and decoupled weight decay of 1e−5, and 1000 warm-up steps with a final learning rate decay to 0.001 of peak, differently to core pre-training. Other settings (batch size, etc) are carried over. We did not find a distillation loss to be helpful here for AFM-on-device, unlike in core pre-training, so the recipe is identical to that used for AFM-server.

No distillation loss

3.2.3 Context lengthening

Finally, we conduct a further 100B tokens of continued pre-training at a sequence length of 32768 tokens, using the data mixture from the continued pre-training stage, augmented with synthetic long-context Q&A data. We also increase the RoPE base frequency from 500k to 6315089, following the scaling laws described in [Liu et al., 2024], with the expectation that this will allow for better short-to-long generalization—which is desirable given that the majority of our pre-training data is comprised of documents that are significantly shorter than 32k tokens long. The recipe is similar to that used for continued pre-training. We examine the long-context performance of AFM-server in Appendix D.

3.2.4 Optimizer

We choose to use a variant of RMSProp [Hinton, 2012] with momentum for AFM pre-training. In particular, we divide the raw gradient by the square-root of a bias-corrected exponential moving average of the squared gradient to produce an instantaneous update, which is clipped to a maximum norm of 1.0 per parameter block, before then further smoothing this estimate over steps with an exponential moving average without bias-correction to produce the net update. Unless otherwise noted, the smoothing constants for both the squared gradient (β_2) and the update (β_1) are set to 0.95. A small constant $\epsilon = 1e{-}30$ is added to the instantaneous squared gradient prior to smoothing, for numerical stability.

The smoothed updates are scaled by the learning rate, weight-decay is added, and then scheduled decay is applied to form the final weight delta. As an additional guard for stability, prior to the optimizer we clip the global gradient norm to 1.0. For a recipe ablation against a more typical configuration, see Appendix A.

3.3 Training infrastructure

The AFM models are pre-trained on v4 and v5p Cloud TPU clusters with the AXLearn framework [Apple, 2023], a JAX [Bradbury et al., 2018] based deep learning library designed for the public cloud. Training is conducted using a combination of tensor, fully-sharded-data-parallel, and sequence parallelism, allowing training to scale to a large number of model parameters and sequence lengths at high utilization. This system allows us to train the AFM models efficiently and scalably, including AFM-on-device, AFM-server, and larger models.

AFM-server was trained on 8192 TPUv4 chips provisioned as 8×1024 chip slices, where slices are connected together by the data-center network (DCN) [Chowdhery et al., 2022]. Only data-parallelism crosses the slice boundary, other types of state sharding are within-slice only as the within-slice interconnect bandwidth is orders of magnitude higher than the DCN. The sustained model-flop-utilization (MFU) for this training run was approximately 52%. AFM-on-device was trained on one slice of 2048 TPUv5p chips.

4 Post-Training

While Apple Intelligence features are powered through adapters on top of the base model (see Section 5 for a deep-dive on the adapter architecture), empirically we found that improving the general-purpose post-training lifts the performance of all features, as the models have stronger capabilities on instruction following, reasoning, and writing.

We conduct extensive research in post-training methods to instill general-purpose instruction following and conversation capabilities to the pre-trained AFM models. Our goal is to ensure these model capabilities are aligned with Apple's core values and principles, including our commitment to protecting user privacy, and our Responsible AI principles. Our post-training efforts include a series of data collection and generation, instruction tuning, and alignment innovations. Our post-training process contains two stages: supervised fine-tuning (SFT) and reinforcement learning from human feedback (RLHF). We present two new post-training algorithms: (1) a rejection sampling fine-tuning algorithm with teacher committee (iTeC), and (2) a reinforcement learning from human feedback (RLHF) algorithm with mirror descent policy optimization and a leave-one-out advantage estimator (MDLOO) that are used on our reinforcement learning iterations and lead to significant model quality improvements.

4.1 Data

We use a hybrid data strategy in our post-training pipeline, which consists of both human annotated and synthetic data. Throughout our data collection and experiment process, we have found data quality to be the key to model success and thus have conducted extensive data curation and filtering procedures.

4.1.1 Human annotations

Demonstration data To fuel the instruction fine-tuning of AFM, we collect high-quality human annotated demonstration datasets from various sources. This dialogue-style data consists of both system-level and task-level instructions (a.k.a. prompts), as well as their corresponding responses. Similar to [Zhou et al., 2024], we observe quality to weigh more importantly than quantity in our experiments. As a result, we focus on key data quality criteria including helpfulness, harmlessness, presentation, and response accuracy, in addition to targeting a diverse task distribution covering Apple Intelligence features. To protect user privacy, we take steps to verify no personally identifiable information is present in our data, and we do not include any personal data stored by users with Apple.

Human preference feedback To iteratively improve AFM's capabilities, we further collect human feedback for reinforcement learning. In particular, we instruct human annotators to compare and rank two model responses for the same prompt to collect side-by-side preference labels. In addition, we also use single-side questions to guide this process. These questions inform raters to grade the model response quality of various aspects including instruction following, safety, factuality, and presentation, and we also retain these labels for model training. We emphasize Apple values and standards in the process. Similar to demonstration data, we find data quality to be crucial for feedback data, and thus we iterate data and model qualities jointly to improve them in a unified flywheel.

4.1.2 Synthetic data

In addition to human annotations, we delve into enhancing data quality and diversity through synthetic data generation. Our findings suggest that when guided by our robust reward models, AFMs are capable of generating high quality responses and for some specific domains, these responses are found to be on par with, or even superior to, human annotations. Therefore, we extend our prompt set to increase the diversity and find that those generated responses can benefit AFMs themselves. In the following, we discuss three domains where we generate synthetic data for AFM post-training: mathematics, tool use, and coding.

Mathematics In the field of mathematics, the wide-ranging subjects and difficulty level make it exceptionally resource-intensive for collecting human demonstrations, since it requires expert knowledge from the human writers. It also becomes impractical to solely rely on human-written content as the model continuously improves. As a consequence, exploring the potential of synthetic data becomes essential to effectively address the challenges.

The creation of synthetic data for mathematics involves two primary stages: generating synthetic math problems and producing their corresponding solutions. For math problem synthesis, we employ several "evolution" strategies

Synthetic data generation for Maths
1. Generating Problems
2. Generating Solutions

where a seed set of prompts are transformed into a much larger set of diverse prompts:

Problem rephrase and reversion. Following the approach in [Yu et al., 2023], we prompt AFM to rephrase seed math questions, and curate reverse questions to derive a specific number in a raw problem statement when provided with the final answer.

Problem evolution. Inspired by the instruction evolving technique [Xu et al., 2023], given a seed problem set $\mathcal{D}_{\text{seed}}$ we prompt AFM to generate two distinct sets of math problems, i.e. $F(\mathcal{D}_{\text{seed}}) \xrightarrow{\text{depth}} \mathcal{D}_{\text{depth}}$, and $F(\mathcal{D}_{\text{seed}}) \xrightarrow{\text{breadth}} \mathcal{D}_{\text{breadth}}$. The in-depth evolution enhances instructions by adding complexities while the in-breadth evolution improves the topic coverage. For both $\mathcal{D}_{\text{breadth}}$ and $\mathcal{D}_{\text{depth}}$, we first perform de-duplication with an embedding model, and subsequently prompt LLMs to ensure the coherence and solvability of the math problems. In addition, for $\mathcal{D}_{\text{depth}}$ a difficulty level is assigned and we only select math problems that score above a specified threshold.

With an augmented set of math questions, we then prompt AFM to synthesize N responses with chain-of-thought per question. If the initial seed data has ground truth, they can be used as an "outcome reward signal" to filter synthesized answers. For problems that require less reasoning steps, we observe that a correct final answer often gets associated with correct intermediate steps. If direct answer checking is unsuccessful or ground truth is unavailable, we instead assess the response correctness by querying an LLM judge. We find that the filtered answers, when fed into the training data, boost our models' math capabilities by a large margin.

Tool use We develop tool-use capabilities such as function call, code interpreter, and browsing through a mixture of synthetic and human data. The model capabilities are first bootstrapped with synthetic data, which focuses on single-tool use cases. We then collect human annotations to improve model capabilities that involve multi-tool and multi-step scenarios. We further augment the human curated function call data by mixing the oracle tool with other similar tools to increase the difficulty of tool selection. In addition, we synthesize parallel function call from human curated function call data to enable the new capability and tool intent detection data based on human curated function call and general SFT data to mitigate tool call over-triggering issues.

Coding The generation of a synthetic coding dataset involves a self-instruct method with rejection sampling. This approach enables the model to learn and generate data autonomously. Starting with 71 different programming topics as the seeds, the model is prompted to generate an initial pool of coding interview-like questions. For each question, the model generates a set of unit

tests and a number of potential solutions. We then use an execution-based rejection sampling method to select the best solution. This involves compiling each potential solution with every unit test and executing them. The solution with the highest number of successful executions is chosen. This results in a collection of (question, test cases, solution) triplets. At the end, we validate the quality of the dataset by filtering the triplets using the number of passed unit tests, resulting in 12K high quality triplets used in the SFT.

4.2 Supervised fine-tuning (SFT)

It has been shown [Chung et al., 2024] that scaling multi-task instruction tuning dramatically enhances model performance on a wide variety of tasks. Similarly, we attempt to scale supervised fine-tuning data to achieve a strong base model for subsequent alignment. During SFT, we collect and train models on demonstration data of a given prompt[3]. We carefully select and combine both human data and synthetic data to form a high quality mixture that covers various natural language use cases.

Data selection We establish a series of quality guards of the data before onboarding them for model training, including ratings from in-house human labelers, automatic model-based filtering techniques, and deduplication with text embeddings. We also scale up the mixture size by a variety of synthetic data generation methods, as described in Section 4.1.2, and rejection sampling as described in Section 4.3.2.

Tuning the mixture ratio In order to tune the mixture weight, we treat it as an optimization problem. Specifically, given a set of weights $(w_1, w_2, ..., w_n)$ where w_i represents the ratio of a specific component in the mixture, we train a model with $w_i \rightarrow w_i \pm \Delta w_i$ and evaluate the quality change on a set of benchmarks. We find that extensively running such experiments can effectively identify the best mixture and remove the least impactful data components.

Training hyperparameters The model is trained with a constant learning rate 5e−6 for AFM-server and 2e−5 for AFM-device models, as well as a drop out rate 0.1. Since the evaluation metrics fluctuate across different checkpoints, we run checkpoint selection based on automatic evaluation benchmarks and best-of-N selection with reward models to test the headroom for RL.

4.3 Reinforcement learning from human feedback (RLHF)

We further use reinforcement learning with collected human preference data to improve model performance and quality. This involves training a robust reward model and applying it in two algorithms of iTeC and MDLOO that we discuss below. We describe more details of our RLHF pipeline in Appendix E.

[3]A prompt may consist of the most recent user instruction as well as all previous user-model-system interactions.

4.3.1 Reward modeling

We train reward models using the human preference data collected with the method in Section 4.1.1. Each human preference data item contains one prompt and two responses along with human labels including:

- The preferred response between the two and the preference level, i.e., whether the preferred response is significantly better, better, slightly better, or negligibly better than the rejected response.
- The single-sided grading of each response, measuring the instruction following property, the conciseness, truthfulness, and harmlessness of each of the responses.

Our reward model training follows the standard practice of reward modeling in RLHF with two main innovations:

- We design a soft label loss function that takes the level of human preference into account.
- We incorporate single-sided gradings as regularization terms in reward modeling.

We employ the commonly used Bradley-Terry-Luce (BTL) model [Bradley and Terry, 1952] for reward modeling in RLHF. In this model, the probability that a human annotator prefers one response over another is modeled as the sigmoid function of the difference of the rewards. Our *soft label* loss function encourages that this probability is high when the preference level is high, e.g., when one response is significantly better than the other, and vice versa. We note that this is different from the *margin*-based loss function in Llama 2 [Touvron et al., 2023], which also leverages the preference level. Empirically, we find that our method works better than the margin-based loss function. Moreover, we also find that using the single-sided gradings as regularization terms can effectively improve the accuracy of the reward model. More details of our reward modeling techniques can be found in Section E.1.

$P(h>l) \propto \sigma(r_h - r_l)$

Llama-2 has margin-based loss function.

4.3.2 Iterative teaching committee (iTeC)

To fully unlock the ability of our model with multiple rounds of RLHF, we propose a novel iterative RLHF framework which effectively combines various preference optimization algorithms, including rejection sampling (RS), Direct Preference Optimization (DPO) [Rafailov et al., 2024] and its variants such as IPO [Azar et al., 2024], and online reinforcement learning (RL). This enables us to bring the benefit of RLHF to AFM models across all sizes and improve their alignment at the same time.

Needs a TON of compute

Iterative committee One of the most important lessons we learned from developing AFM RLHF is to refresh online human preference data collection using a diverse set of the best performing models. Specifically, for each batch

of human preference data collection, we set up a collection of latest promising models trained from SFT, RS, DPO/IPO, and RL, as well as best models from the previous iterations, which we refer to as "model committee". We collect pairwise human preference on responses sampled from the latest model committee.

After acquiring each batch of human preference data, we refresh our reward model, and further train a new set of models using the collection of preference optimization algorithms. We then continue the next round of iterative RLHF data collection with a new model committee.

Committee distillation We further run rejection sampling (distillation) from the model committee with the latest reward model as a reranker. Instead of reranking at global-level, i.e., picking a single best performing model from the committee and using it as a teacher model, we rerank model responses at the prompt-level. Specifically, for each prompt, we sample multiple responses from each model in the committee, and use the latest reward model to select the best response for each prompt. This allows us to combine the advantages of models trained by different preference optimization algorithms. For instance, we find that algorithms that leverage negative examples, e.g., online RLHF, DPO, IPO, to be better in improving reasoning skills such as math, while rejection sampling fine-tuning learns instruction following and writing skills more effectively.

* Algos that use negative examples like DPO, oRLHF
=> Good at reasoning, math

* Rejection Sampling
=> Good at inst. following and writing skills.

Scaling up distillation In order to bring the RLHF improvements to AFM models across all sizes, we scale up distillation from the model committee. Different from larger models, where carefully iterating data and model quality matters much more than data quantity, we find smaller models can achieve tremendous improvement when we scale up the number of prompts for distillation. Our final AFM-on-device model is trained on more than 1M high quality responses generated from the model committee.

Smaller LMs
↳ Quantity sometimes more important than Quality.

4.3.3 Online RLHF algorithm: MDLOO

In this section, we introduce our online reinforcement learning algorithm MDLOO, where we decode responses during model training and apply RL algorithms to maximize the reward.

We use the commonly adopted RLHF objective that maximizes the KL-penalized reward function [Ouyang et al., 2022]:

$$\max_{\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x)} \left[r_\phi(x, y) - \beta D_{\mathrm{KL}} \left(\pi_\theta(\cdot|x) \| \pi_{\mathrm{ref}}(\cdot|x) \right) \right], \tag{1}$$

where $\mathcal{D}$ is the prompt distribution, $D_{\mathrm{KL}}(\cdot\|\cdot)$ denotes the Kullback-Leibler divergence between two distributions, and β is the coefficient that controls the divergence between the behavior policy π_θ and a reference policy π_{ref}, which is typically a model trained by SFT. In our RL training, we use the reward

function

$$R(x, y) = r_\phi(x, y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}, \tag{2}$$

whose expectation is equivalent to Equation 1. We consider the bandit setting where the generation of the entire response is considered as one action, and we do not use the value network (a.k.a. the critic) to obtain the per-token reward or advantage.

No per-token reward !!

Similar to commonly used RLHF algorithms such as PPO [Schulman et al., 2017], we use a trust-region based policy iteration algorithm. We made two main design choices in our online RL algorithm:

- We use the Leave-One-Out (LOO) estimator to estimate the advantage of a prompt-response pair, similar to a recent work [Ahmadian et al., 2024].
- We use Mirror Descent Policy Optimization (MDPO) [Tomar et al., 2020] to optimize the policy, differently from the more commonly used clipping-based PPO method.

Thus, we name our online RL algorithm *Mirror Descent with Leave-One-Out estimation* (MDLOO). More specifically, during the decoding stage of the algorithm, we decode multiple responses for each prompt, and assign the *advantage* of each response to be the difference of the reward of the (prompt, response) pair and the mean reward of the other responses generated by the same prompt. Intuitively, this estimator aims to measure how much better a response is compared to a typical response. Empirically, we find this advantage estimator crucial for stabilizing the RL algorithm and achieving strong results. Moreover, we use a KL-regularization-based trust region method, i.e. MDPO, to control the policy change in each iteration. We find that this algorithm is more effective than PPO in our setting. More details of our online RLHF algorithm can be found in Section E.2.

5 Powering Apple Intelligence features

Our foundation models are designed for Apple Intelligence, the personal intelligence system integrated into supported models of iPhone, iPad, and Mac. We have built these models to be fast and efficient. And while we have achieved impressive levels of broad capability in our base model, the actual relevant measure of its quality is how it performs on specific tasks across our operating systems.

Here we have found that we can elevate the performance of even small models to best-in-class performance through task-specific fine-tuning and have developed an architecture, based on runtime-swappable adapters, to enable the single foundation model to be specialized for dozens of such tasks. A high-level overview is presented in Figure 2.

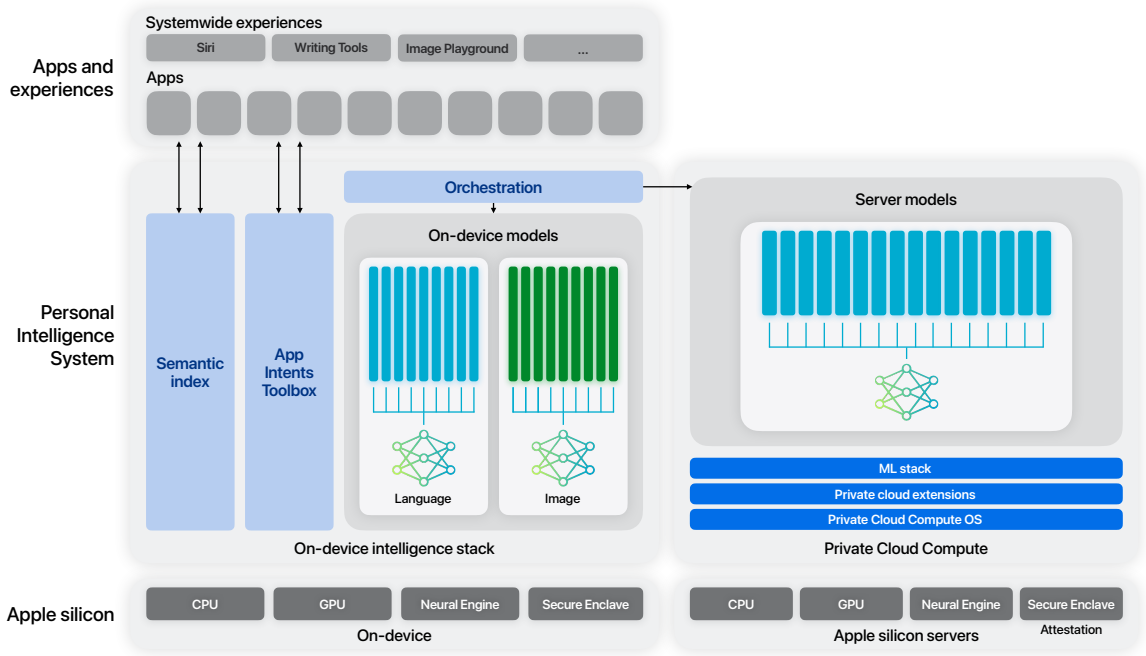

Figure 2: Architecture of Apple Intelligence with adapters for the language on-device and server models and the image models. In this report we are only describing the text models.

5.1 Adapter architecture

Our foundation models are fine-tuned for users' everyday activities, and can dynamically specialize themselves on-the-fly for the task at hand. We use LoRA [Hu et al., 2021] adapters, small neural network modules that can be plugged into various layers of the base model, to fine-tune our models for specific tasks. For each task, we adapt all of AFM's linear projection matrices in the self-attention layers and the fully connected layers in the pointwise feedforward networks. By fine-tuning only the adapters, the original parameters of the base pre-trained model remain unchanged, preserving the general knowledge of the model while tailoring the adapters to support specific tasks.

We represent the values of the adapter parameters using 16 bits, and for the ∼3 billion parameter on-device model, the parameters for a rank 16 adapter typically require 10s of megabytes. The adapter models can be dynamically loaded, temporarily cached in memory, and swapped—giving our foundation model the ability to specialize itself on the fly for the task at hand while efficiently managing memory and guaranteeing the operating system's responsiveness.

To facilitate the training of the adapters, we created an efficient infrastructure that allows us to rapidly add, retrain, test, and deploy adapters when the base model or the training data gets updated or new capabilities are

required. It is worth noting that the adapter parameters are initialized using the accuracy-recovery adapter introduced in Section 5.2.

5.2 Optimizations

The AFM models are designed to support our users throughout their daily activities, and both inference latency and power efficiency are important for the overall user experience. We apply various optimization techniques to allow AFM to be efficiently deployed on-device and in Private Cloud Compute. These techniques significantly reduce memory, latency, and power usage while maintaining the overall model quality.

In order to fit AFM into a constrained memory budget of edge devices and reduce inference cost, it is critical to apply model quantization techniques to reduce the effective bits per weight while maintaining the model quality. Previous works have found that 4-bit quantized models only have marginal loss of quality (typically measured in pre-training metrics) compared to the original 32/16-bit float-point versions. Since AFM is expected to support a diverse set of product features, it is essential that the quantized model retains capabilities in specific domains critical to these use cases. To achieve an optimal trade-off between model capacity and inference performance, we have developed state-of-the-art quantization methods and a framework that utilizes accuracy-recovery adapters. This allows us to achieve near-lossless quantization that is on average less than 4 bit-per-weight, and provides flexible quantization scheme choices.

Methods The model is compressed and quantized, on average under 4-bit-per-weight, after the post-training stages (details of the quantization scheme will be discussed later). The quantized model often shows a moderate level of quality loss. Therefore, instead of directly passing the quantized model to application teams for feature development, we attach a set of parameter-efficient LoRA adapters for quality recovery. We make sure that these LoRA adapters training recipes are consistent with pre-training and post-training processes. Then, products will fine-tune their own feature-specific LoRA adapters by initializing the adapter weights from the accuracy-recovery adapters, while keeping the quantized base model frozen.

First, pretraining of LoRA adapters is done to recover pay loss from quantization

It is noteworthy that training accuracy-recovery adapters is sample-efficient and can be considered as a mini-version of training the base model. During the pre-training stage of the adapters, we only require approximately 10 billion tokens ($\sim$ 0.15% of base model training) to fully recover the capacity for the quantized model. Since application adapters will fine tune from these accuracy-recovery adapters, they do not incur any additional memory usage or inference costs. Regarding adapter size, we found that adapter rank of 16 offers the optimal tradeoff between model capacity and inference performance. However, to provide flexibility for various use cases, we provide a suite of accuracy-recovery adapters in different ranks $\{8, 16, 32\}$ for application teams to select from. In Appendix F, we provide detailed evaluation results across

unquantized, quantized, and accuracy-recovered models and show that the recovered models perform much closer to the unquantized version.

Quantization schemes Another benefit brought by accuracy-recovery adapters is that they allow more flexible choices of quantization schemes. Previously when quantizing LLMs, people typically group the weights into small blocks, normalize each block by the corresponding maximal absolute values to filter out outliers, then apply quantization algorithms in a block basis. While a larger block size yields lower effective bits per weight and a higher throughput, the quantization loss would increase. In order to balance this tradeoff, it is common to set block size to a small value, like 64 or 32. In our experiments, we found that accuracy-recovery adapters can greatly improve the pareto frontier in the tradeoff. More errors will be recovered for more aggressive quantization schemes. As a result, we are able to use a highly-efficient quantization scheme for AFM without worrying about losing model capacity. Specifically, our AFM-on-device model running on Apple Neural Engine (ANE) uses palettization: for projection weights, every 16 columns/rows share the same quantization constants (i.e., lookup tables) and are quantized using K-means with 16 unique values (4-bit). The quantization block size can be up to 100k. Besides, since AFM's embedding layer is shared between the input and output, it is implemented differently from projection layers on ANE. Hence, we quantize the embedding using per-channel quantization with 8-bit integers for better efficiency.

Mixed-precision quantization Residual connections exist in every transformer block and every layer in AFM. So it is unlikely that all layers have the equal importance. Following this intuition, we further reduce the memory usage by pushing some layers to use 2-bit quantization (default is 4-bit). On average, AFM-on-device can be compressed to only about 3.5 bits per weight (bpw) without significant quality loss. We choose to use 3.7 bpw in production as it already meets the memory requirements.

Interactive model analysis We use an interactive model latency and power analysis tool, Talaria [Hohman et al., 2024], to better guide the bit rate selection for each operation.

More discussions The usage of quantized model and LoRA adapters look conceptually similar to QLoRA [Dettmers et al., 2024]. While QLoRA was designed to save computational resources during fine-tuning, our focus is on the ability to switch between different LoRA adapters to efficiently support high performance across various specific use cases. Before feature-specific finetuning, we first train accuracy-recovery adapters on the same pretraining and post-training data, which is critical to preserve the model quality. The accuracy-recovery framework can be combined with different quantization techniques, like GPTQ [Frantar et al., 2022] and AWQ [Lin et al., 2024], since it does not depend directly on the quantization method itself. The feature

adapters described in Section 5 are initialized from these accuracy-recovery adapters.

5.3 Case study: summarization

We use the AFM-on-device model to power summarization features. We worked with our design teams to create specifications for summaries of Emails, Messages, and Notifications.

While AFM-on-device is good at general summarization, we find it difficult to elicit summaries that strictly conform to the specification. Therefore, we fine tune a LoRA adapter on top of the quantized AFM-on-device for summarization. The adapter is initialized from the accuracy-recovery adapter as described in Section 5.2. We use a data mixture consisting of input payloads covering Emails, Messages, and Notifications. These payloads include public dataset, vendor data, and internally generated and submitted examples. All the data have been approved to use for production. Vendor data and internally generated data have been anonymized to remove the user information. Given these payloads, we generated synthetic summaries using AFM-server according to product's requirements. These payloads and summaries are used for training.

Synthetic summaries We use AFM-server to generate synthetic summaries. We apply a series of rule-based filters followed by model based filters. Rule-based filters are based on heuristics such as length constraints, formatting constraints, points of view, voice, etc. Model-based filters are used to screen more challenging problems such as entailment. Our synthetic data pipeline allows us to efficiently generate a large amount of training data and filter it out by an order of magnitude to retain high-quality examples for fine tuning.

Prompt injection We find that AFM-on-device is prone to following instructions or answering questions that are present in the input content instead of summarizing it. To mitigate this issue, we identify a large set of examples with such content using heuristics, use AFM-server to generate summaries, as it does not exhibit similar behavior, and add this synthetic dataset to the fine tuning data mixture.

Cool!! Gemini miss this in their initial iterations

6 Evaluation

We evaluate the AFM models on pre-training (Section 6.1), post-training (Section 6.2), and most importantly, feature-specific (Section 6.3) benchmarks.

6.1 Pre-training evaluation

In this section we present common few-shot pre-training evaluation metrics. While these benchmarks are useful for tracking our progress on pre-training, we found that human evaluations on the post-trained models (Section 6.2) and

feature adapters (Section 6.3) are more closely correlated to end-to-end user experience.

We evaluate AFM pre-trained models with common open-sourced evaluation harnesses and benchmarks. Table 2 presents the results of AFM-on-device and AFM-server on HELM MMLU v1.5.0 [Liang et al., 2023], which tests 5-shot multiple-choice question answering across 57 subjects. Also see Table 3 and Table 4 for the results of AFM-server on a subset of the HuggingFace OpenLLM leaderboard V1 [Huggingface, 2024] and the HELM-Lite v1.5.0 benchmark suite [Stanford, 2024], respectively. These benchmarks show that the AFM pretrained models possess strong language and reasoning capabilities and provide a solid foundation for post-training and feature fine-tuning.

	AFM-on-device	**AFM-server**
MMLU (5 shot)	61.4	75.4

Table 2: HELM MMLU-5s [Liang et al., 2023] v1.5.0 evaluation results.

	AFM-server
MMLU (5-shot)	75.3
GSM8K (5-shot)	72.4
ARC-c (25-shot)	69.7
HellaSwag (10-shot)	86.9
Winogrande (5-shot)	79.2

Table 3: A subset of Open LLM Leaderboard [Huggingface, 2024] V1 evaluation results.

6.2 Post-training evaluation

We evaluate post-training models on comprehensive benchmarks and compare AFM models with various open-source models, as well as GPT-3.5 and GPT-4.[1] All results reported in this section are obtained using AFM-on-device and AFM-server base models without any adapter, in `bfloat16` precision. In this section, we first present human evaluation results that measure the AFMs general capabilities, and then present results for several specific capabilities and domains.

[1]We compared against the following model versions: gpt-3.5-turbo-0125, gpt-4-0125-preview, Gemini-1.5-Pro-0514, DBRX Instruct, Phi-3-mini-4k-instruct, LLaMA 3 8B Instruct, LLaMA 3 70B Instruct, Mistral-7B-Instruct-v0.2, Mixtral-8x22B-Instruct-v0.1, Gemma-1.1-2B, and Gemma-1.1-7B.

	AFM-server
Narrative QA	77.5
Natural Questions (open)	73.8
Natural Questions (closed)	43.1
Openbook QA	89.6
MMLU	67.2
MATH-CoT	55.4
GSM8K	72.3
LegalBench	67.9
MedQA	64.4
WMT 2014	18.6

Table 4: HELM-Lite v1.5.0 [Stanford, 2024] pre-training evaluation results. N.B. Many benchmarks (e.g. MMLU) differ significantly from commonly used settings.

6.2.1 Human evaluation

Human evaluation simulates practical use cases and user feedback, and so often serves as the gold standard for language model evaluation. Consequently, we conduct extensive human evaluations both while developing the model and to evaluate its final form. We collect sets of evaluation prompts to test the model on different aspects, including both general capabilities and safety. For each prompt, two model responses are presented to human raters anonymously for side-by-side comparisons. Depending on the nature of the evaluation, a detailed guideline containing grading principles and examples of single-response ratings and side-by-side preference ratings is provided to human raters to ensure consistent grading standards and evaluation quality. Each pair of model responses is graded by multiple graders and their ratings are aggregated for final results. Overall, we find human evaluation to align better with user experience and provide a better evaluation signal than some academic benchmarks that use LLMs as graders. In this section, we present the results for human evaluation on general capabilities, and the safety evaluation results are provided in Section 7.6.

We collect a comprehensive set of 1393 prompts to evaluate the general model capabilities. These prompts are diverse across different difficulty levels and cover major categories including: analytical reasoning, brainstorming, chatbot, classification, closed question answering, coding, extraction, mathematical reasoning, open question answering, rewriting, safety, summarization, and writing. To prevent overfitting, when preparing training data, we conduct decontamination against our evaluation prompts.

In Figure 3, we compare AFM with both open-source models (Phi-3, Gemma-1.1, Llama-3, Mistral, DBRX-Instruct) and commercial models (GPT-3.5, and GPT-4). AFM models are preferred by human graders over competitor models. In particular, AFM-on-device obtains a win rate of 47.7% when

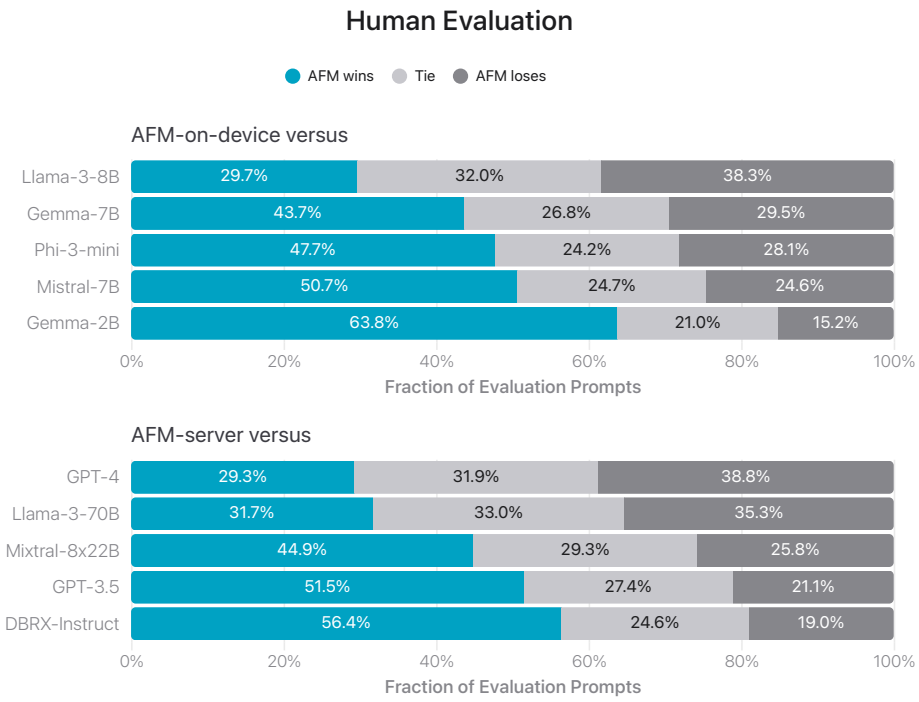

Figure 3: Side-by-side evaluation of AFM-on-device and AFM-server against comparable models. We find that our models are often preferred over competitor models by human graders.

compared to Phi-3-mini despite being 25% smaller in model sizes, and even outperforms open-source strong baselines Gemma-7B and Mistral-7B that are more than twice larger in the number of parameters. When compared to closed-source models, AFM-server achieves competitive performance, scoring a win rate of more than 50% and a tie rate of 27.4% against GPT-3.5.

6.2.2 Instruction following

Instruction following (IF) is the core capability we desire of language models, as real-world prompts are often sophisticated and contain complex instructions. We emphasize the importance of instruction following in both our RLHF data collection and human evaluation. In this subsection, we evaluate our models' IF skills using automated benchmarks.

In Figure 4 we evaluate AFM-on-device and AFM-server on the public IFEval benchmark [Zhou et al., 2023], respectively. This benchmark measures a language model's capability to generate responses that precisely follow instructions in the prompt. The instructions in this benchmark typically include requirements on the response length, format, content, etc. We find AFM-on-device and AFM-server to achieve superior performance on both instruction-level and prompt-level accuracy. In addition, we also benchmark AFM models on the AlpacaEval 2.0 LC benchmark [Dubois et al., 2024] to

measure general instruction-following capability, and results suggest that our models are highly competitive.

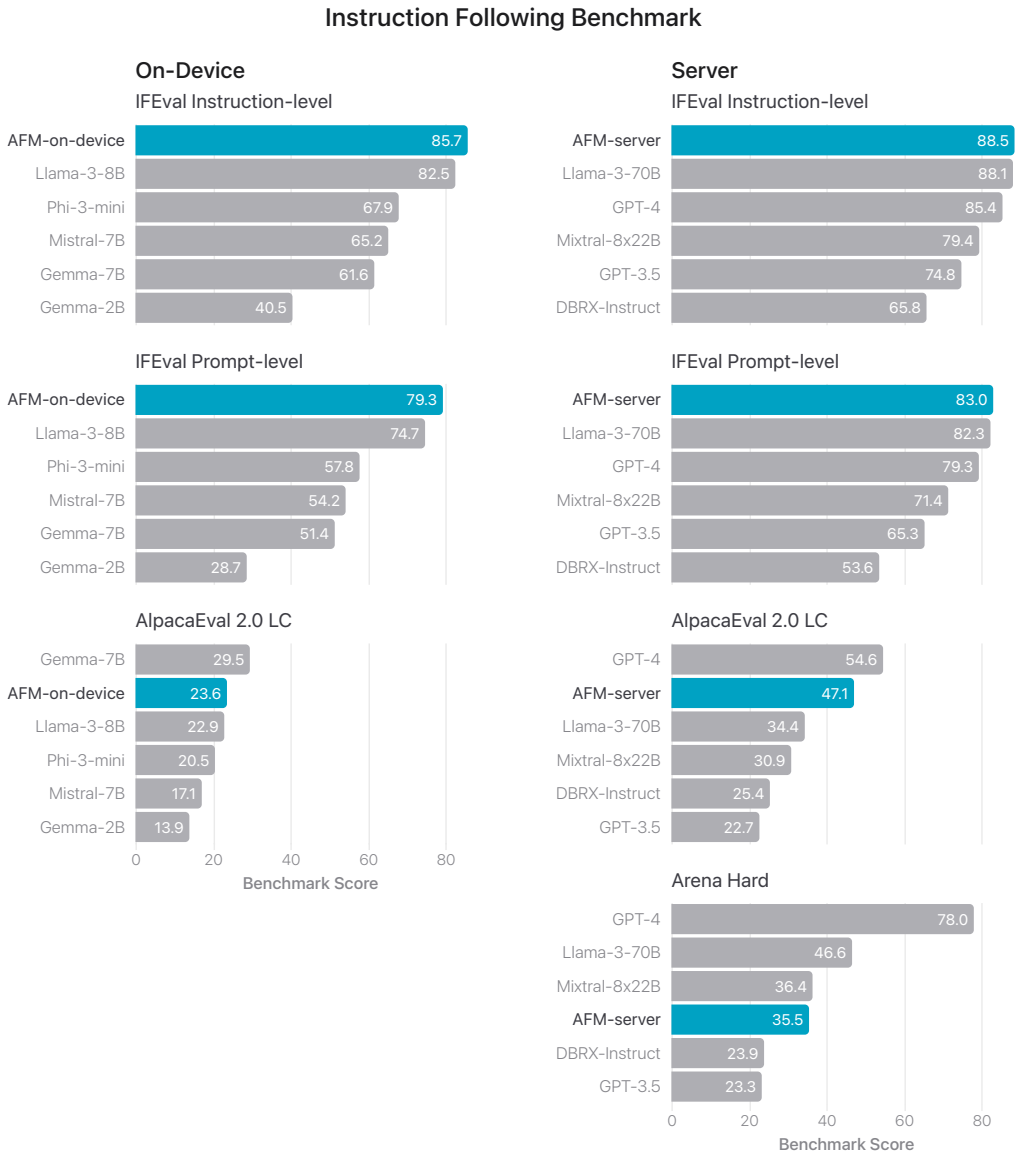

Figure 4: Instruction-following capability (measured with IFEval) for AFM models and relevant comparison models (higher is better). The AlpacaEval 2.0 LC results for Mistral 7B, Llama3 8B, Llama3 70B, DBRX-Instruct, and Mixtral 8x22B are obtained from the AlpacaEval leaderboard [Taori et al., 2023]. The Arena Hard results for comparison models are from the Arena-Hard-Auto leaderboard [Li et al., 2024b]. All other results are from our own evaluations.

6.2.3 Tool use

In tool use applications, given a user request and a list of potential tools with descriptions, the model can choose to issue tool calls by providing a structured output specifying the name and parameter values of the tools to call. We expect the tool descriptions to follow the OpenAPI specification.[4]

We evaluate on the public Berkeley Function Calling Leaderboard benchmarks [Patil et al., 2023] via native support of function calling, using the AST metrics.

As shown in Figure 5, AFM-server achieves the best overall accuracy, outperforming Gemini-1.5-Pro-Preview-0514 and GPT-4.

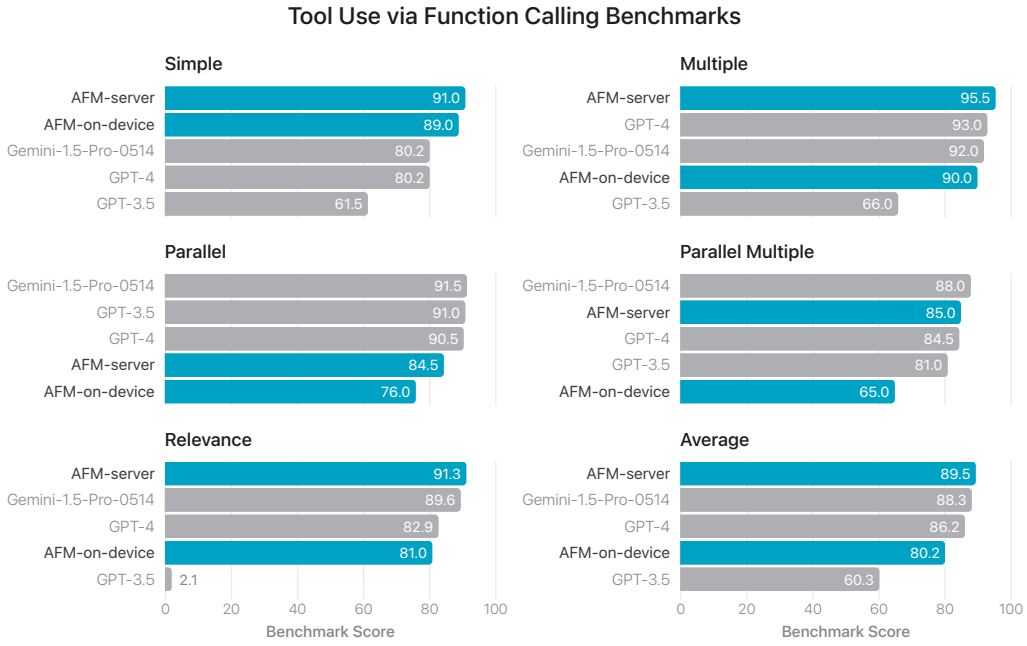

Figure 5: Berkeley Function Calling Leaderboard Benchmark evaluation results on Function Calling API, along-side relevant sampled comparisons. Numbers were collected from the Gorilla leaderboard [Patil et al., 2023].

6.2.4 Writing

Writing is one of the most critical abilities for large language models to have, as it empowers various downstream use cases such as changing-of-tone, rewriting, and summarization. However, assessing writing quality is a non-trivial task, and not well-covered in the above public benchmarks.

We evaluate AFM's writing ability on our internal summarization and composition benchmarks, consisting of a variety of writing instructions. Following LLM-as-a-judge [Zheng et al., 2024], we design a grading instruction

[4] https://github.com/OAI/OpenAPI-Specification

for each summarization and composition task, and prompt GPT-4 Turbo to assign a score from 1 to 10 for model responses.[5] We note that there are certain limitations and biases associated with using an LLM as a grader, such as length bias.

We compare AFM with a few of the most outstanding models, along with smaller-scale open-source models. As shown in Figure 6, AFM-on-device can achieve comparable or superior performance when compared to Gemma-7B and Mistral-7B. AFM-server significantly outperforms DBRX-Instruct and GPT3.5 and is comparable to GPT4.

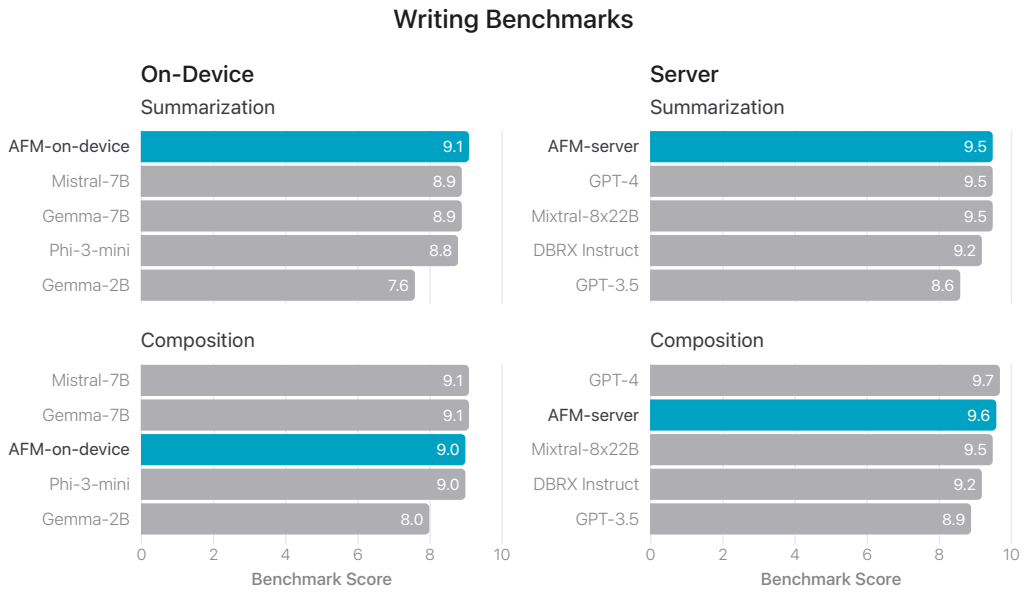

Figure 6: Writing ability on internal summarization and composition benchmarks (higher is better) for AFM-on-device and AFM-server alongside relevant sampled comparisons. We find that our models perform better or similar to related models.

6.2.5 Math

In Figure 7, we compare post-training AFM's performance on math benchmarks including GSM8K [Cobbe et al., 2021] and MATH [Hendrycks et al., 2021]. We use 8-shot chain-of-thought (CoT) [Wei et al., 2022] prompt for GSM8K and 4-shot CoT prompt [Lewkowycz et al., 2022] for MATH. We conduct all evaluations using an internal automated evaluation pipeline. We see that the AFM-on-device significantly outperforms Mistral-7B and Gemma-7B, even at less than half of their sizes.

[5]Due to the choice of using GPT-4 as judge, the score of GPT-4 Turbo can be overestimated.

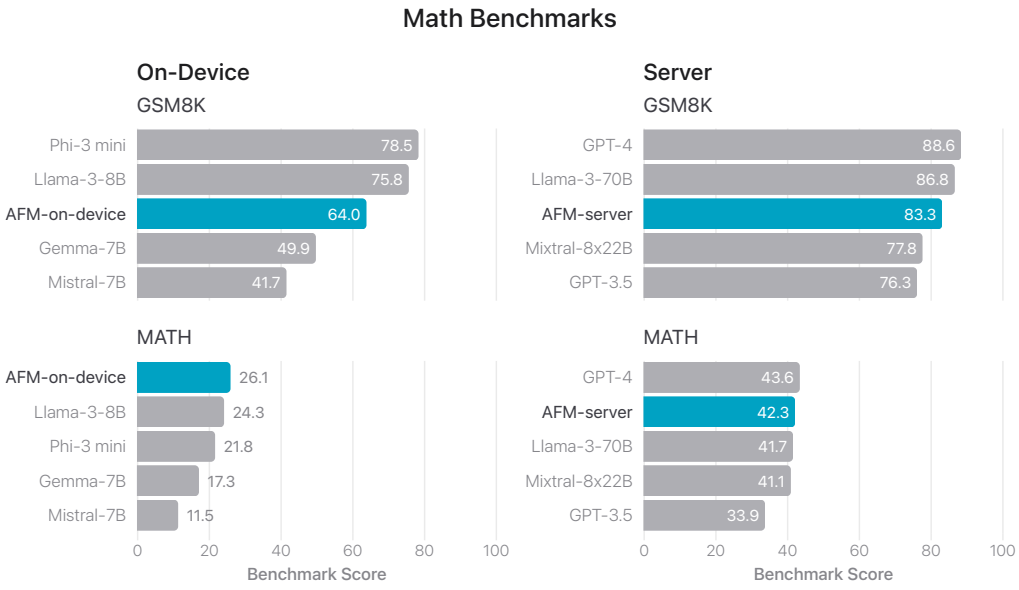

Figure 7: Math benchmarks for AFM-on-device and AFM-server alongside relevant sampled comparisons. GSM8K is 8-shot and MATH is 4-shot. All results are collected with an internal automated evaluation pipeline.

6.3 Summarization feature evaluation

The product team specifications for summarizing Emails, Messages, and Notifications necessitated a tailor-made set of guidelines, metrics, and specialized graders to evaluate summarization quality against various open-source, licensed, and proprietary datasets.

Datasets. We sampled abundant payloads carefully for each use case. These evaluation datasets emphasize a diverse set of inputs which our product features are likely to face in production, and include a stratified mixture of single and stacked documents of varying content types and lengths. We developed a pipeline to build evaluation datasets that simulate real user inputs.

Graders. We enlisted a pool of highly-trained, full-time, Apple-employed human graders with specialized writing and comprehension skills to evaluate summarization quality. To qualify for grading projects, each grader must pass a series of eligibility and training steps, which include a required bachelor's degree in a writing-related discipline, customized training sessions, and consistently high performance against internal grading quality benchmarks.

Grading guidelines. During the evaluation task, graders are presented with a specification for the summary, the original input content, and the output summary. Graders assess the summary on each the following sub-dimensions of quality using 3 point scales ("good", "neutral", or "poor"):

Composition: Evaluates the overall readability of the summary considering grammar, punctuation, spelling, and brevity.

Comprehensiveness: Evaluates how comprehensive the summary is in capturing the essential points or calling out any actions/conclusions for the user.

Groundedness: Evaluates how grounded the summary is with respect to the original payload. Summaries that are not completely grounded may contain details that are exaggerated, inferred, inaccurate, or hallucinated.

Following instructions: Evaluates whether the summary meets specific style and formatting requirements. Requirements are tailored to each feature and reflect specific product and design expectations.

Harmfulness: Evaluates whether the summary contains content that is harmful or unsafe according to Apple's safety taxonomy.

A summary is classified as "poor" if *any* of the sub-dimensions are "poor" according to predefined product specifications. Likewise a summary is "good" only if *all* sub-dimensions are good. These classifications are used to compute "Good/Poor Result Ratio" metrics defined as the percentage of good/poor summaries out of all summaries.

Results. We ask human graders to evaluate the summarization quality of the AFM-on-device adapter, Phi-3-mini, Llama-3-8B, and Gemma-7B. Figure 8 shows that AFM-on-device-adapter overall outperforms the other models.

7 Responsible AI

7.1 Overview

Apple Intelligence is developed responsibly and designed with care to empower our users, represent them authentically, and protect their privacy. Of primary importance to our Responsible AI approach is that we are ultimately delivering intelligent, well-defined tools that address specific user needs. Having a clear definition of what a feature is intended to do allows us to better identify any potential safety gaps.

We have developed a safety taxonomy in order to be comprehensive and consistent in the design and evaluation of our generative AI-powered features. This taxonomy builds and extends Apple's extensive experience in using artificial intelligence and machine learning to deliver helpful features to users around the world, and is updated regularly as we develop and test features. Currently, it consists of 12 primary categories comprised of 51 subcategories, including "Hate Speech, Stereotypes, and Slurs", "Discrimination, Marginalization, and Exclusion", "Illegal Activities", "Adult Sexual Material", and "Graphic Violence."

The taxonomy serves as a structured way to consider potential issues and risks relative to each specific feature. As new or additional risks are identified, we develop and revise the associated policies that are contextualized to each

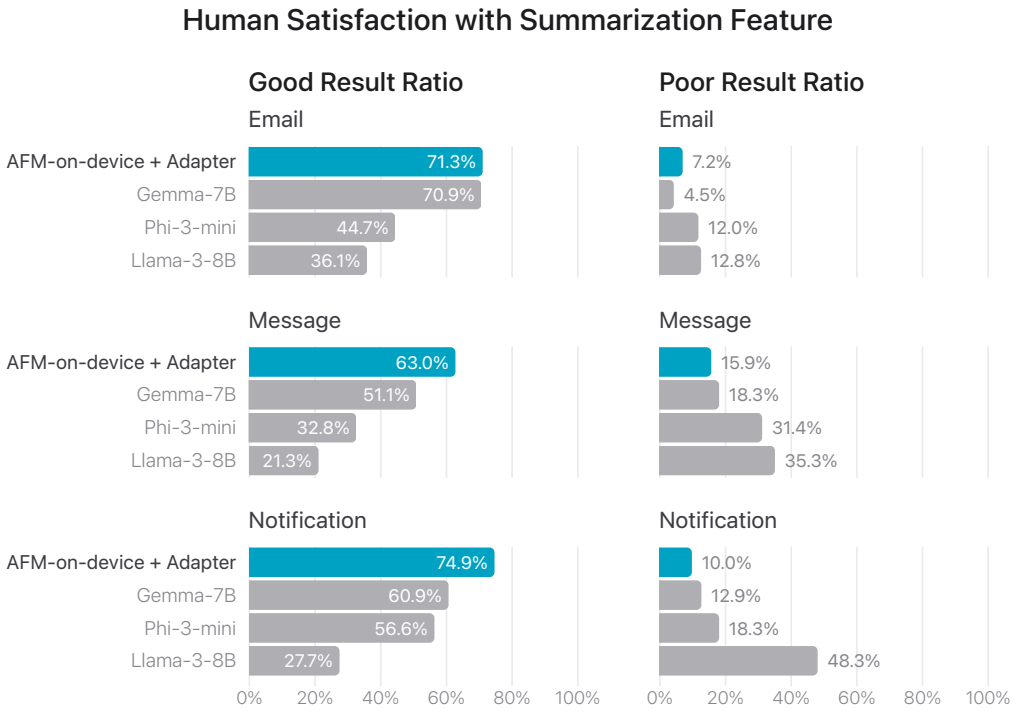

Figure 8: Ratio of "good" and "poor" responses for three summarization use cases relative to all responses. Summaries are classified as "good", "neutral", or "poor" along five dimensions. A result is classified as "good" if *all* of the dimensions are good (higher is better). A result is classified as "poor" if *any* of the dimensions are poor (lower is better). Overall, our AFM-on-device adapter generates better summaries than comparable models.

individual feature, taking into account the specific needs that it serves, the content it produces, and the appropriate mitigations. They are developed with extensive internal and external input from academics, AI ethicists, trust and safety, and legal experts to better identify and understand the relevant risks, the potential severity of such risks, and the potential disparate impact these risks may have on certain groups. These policies guide our work in data collection, human annotation, model training, guardrails development, evaluation, and red teaming.

Particularly, the taxonomy is not itself the sole determinant of our policy. For example, content that may fall within the safety taxonomy is not necessarily always blocked, as doing so unilaterally may be in conflict with other aspects of Apple's Responsible AI development principles, such as "respecting how our users choose to use these tools to accomplish their goals." Thus, features that operate as tools may be more permissive in the kinds of content they operate over and produce in order to effectively address the user's intent. On the other hand, features that may generate content beyond a user's specified intent may need to be more constrained. Regardless, we strive for some categories of harm

to always be treated with special care (such as any content that relates to self harm) while other categories will always be blocked (such as illegal content).

In addition, our Responsible AI principles are built into every stage of Apple Foundation Models and Apple Intelligence as well as the safety taxonomy, which helps us evaluate risks and formulate policies feature by feature. We include safety-oriented data as part of our fine-tuning of specific adapters tailored by use case. Furthermore, at the time of inference, we also run guardrail models [Inan et al., 2023] as pre- and post-processing steps to evaluate potential harm at both the input and output level. Finally, we have mechanisms in place to continuously and proactively improve our AI tools with the help of ongoing user feedback.

7.2 Pre-Training

At the pre-training stage, we take several steps to ensure that the values as outlined above are upheld. We follow a strict data policy ensuring that no Apple user data is included, as well as conduct rigorous legal review for each component in the training corpus. Further, we perform safety filtering to reduce potentially harmful content, including NSFW content, profanity, spam, and PII or financial data.

Because pre-training is a step which is shared among various downstream features, our safety mitigations aim to retain general capabilities that allow us to iterate on the taxonomy and policy at a per-feature level, without hurting the helpfulness of these downstream models. We take learnings from prior work to avoid overly aggressive filtering at the pre-training stage, which has potential benefits in safety alignment [Touvron et al., 2023]. Intuitively, the pre-trained model should be aware of content that downstream features and policies may require it to handle – in some cases with care, or in other cases operating over such content directly.

7.3 Post-Training

In the post-training phase, we aim to instill a baseline level of alignment with our Responsible AI principles to avoid necessitating the full complexities of post-training (such as RLHF) in each downstream model that builds on top of the foundation model. In doing so, there are two key considerations:

1. We must ensure our models produce output that is helpful to users, while minimizing potential harm.

2. We must contextualize our safety taxonomy and policies on a feature by feature basis to deliver the best possible user experience.

To balance helpfulness and harmlessness trade-off, our solution is to treat safety alignment as one of the many core post-training tasks that are evaluated and iterated on in tandem, instead of as a separate stage of training. Specifically, we include adversarial data into our SFT and RLHF training corpora that is curated according to our policy and values by partnering closely with trusted

vendors. We also incorporate safety tasks and benchmarks into the automatic and human evaluations used during model development.

In total, over 10% of the training data are adversarial or related to safety or sensitive topics, including single and multi-turn safety category annotations, pairwise and overall preference ratings, and annotator rewrites. This data is either used directly or as seed data for synthetic data generation, as described in Section 4.1.2.

We do additional work to achieve appropriate safety behavior for each feature beyond baseline alignment. A primary way that we do this is by collecting safety-specific training data and including it when fine-tuning adapters. For instance, in fine-tuning our summarization adapter we sought to improve aspects such as, improving robustness against malicious questions included within the content to be summarized, and reducing the likelihood that summaries would inadvertently amplify harmful or sensitive content to be summarized.

7.4 Guarding against malicious code

Code generation requires special care. Our code benchmarks involve actually executing the generated code to determine both syntactic and semantic correctness. Thus, responsible training of code models involves treating all generated code as unsafe by default – all code is always executed in a fully locked down environment with no access to the internet or any internal or external services. Specifically, the locked down environment is managed with FireCracker [Agache et al., 2020], with a FireCracker jailer at the cluster level.

7.5 Red teaming

Red teaming attempts to elicit safety policy violating responses from models, or harmful responses for which no policy yet exists. These results inform both policy development as well as the focus and content of safety evaluation datasets. These in turn can influence design, engineering, and shipping readiness decisions.

Red teaming is a fundamentally creative endeavor that requires red teamers to employ combinations of attack vectors to probe known model vulnerabilities, and try to discover new ones. Attack vectors used when engaging with language models include jailbreaks/prompt injections, persuasive techniques [Zeng et al., 2024], and linguistic features known to cause model misbehavior (e.g. slang, code-switching, emojis, typos).

We employ both manual and automatic red-teaming [Ganguli et al., 2022] to elicit potentially unknown failure modes of the aligned models. More recent works [Touvron et al., 2023] suggest that automated processes can potentially generate even more diverse prompts than humans, previously seen as the "gold" standard for data collection. These automated processes can include using the language models themselves to identify gaps, some of which may be unintuitive or even surprising. Such examples can be used directly as synthetic training or evaluation data and to inform future data collection efforts.

A basic human red teaming task schema is as follows: a red teamer is assigned a safety taxonomy category and attack vector(s). They author an input to the model, using that attack vector, that is intended to elicit a response containing content from that category. If the response does not contain the target content, the red teamer can engage in a fixed number of conversational turns, after which they provide a final harmfulness rating of the model output and list the taxonomy categor(ies) in it, if any. To ensure annotation quality, red teamers also provide an overall confidence score for their ratings.

In addition to red teaming at the base model level, we also red team specific features. Red teaming projects at the feature level use feature-specific guidelines with attack vectors informed by the feature's safety policy and engineering concerns. These projects can provide in-depth probing of known risks for that particular feature and also adversarially probe for unknown vulnerabilities.

Our red teaming projects are run using internal and external crowds. To ensure responsible data collection, due to the sensitive nature of red teaming we: 1) make red teaming completely voluntary; 2) impose a strict time limit on how much each red teamer spends on the tasks per week; 3) provide health and well-being resources available around the clock; and 4) maintain an open line of communication with internal red teamers via weekly office hours and a Slack channel for them to communicate any concerns that arise.

7.6 Evaluation

As mentioned in previous sections, safety is one of the many axes iterated on during foundation model development, and therefore undergoes the same automatic and human evaluation cycles during post-training.

Safety evaluation set To reduce noise, cost, and turn-around time during human evaluations, we must ensure that our safety evaluation sets are clean, yet challenging and comprehensive. To that end, we filter out "easy" prompts which consistently yield low harmfulness responses across different versions of the model, and employ an embedding-based analysis to improve our evaluation prompt set coverage. Overall, we curate a set of over a thousand adversarial prompts to test AFM's performance on harmful content, sensitive topics, and factuality according to our safety policy.

Safety evaluation results Figure 9 summarizes the violation rates of different models evaluated by human graders on this safety evaluation set. Lower is better. Both AFM-on-device and AFM-server are robust to adversarial prompts, achieving violation rates significantly lower than open-source and commercial models. In addition, we report side-by-side human preference on our safety evaluation prompts in Figure 10. AFM models are preferred by human graders as safe and helpful responses over competitor models.

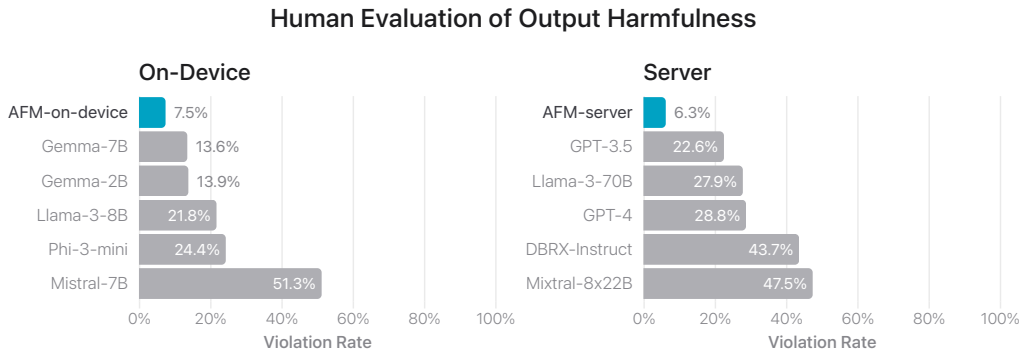

Figure 9: Fraction of violating responses for harmful content, sensitive topics, and factuality (lower is better). Our models are robust when faced with adversarial prompts.

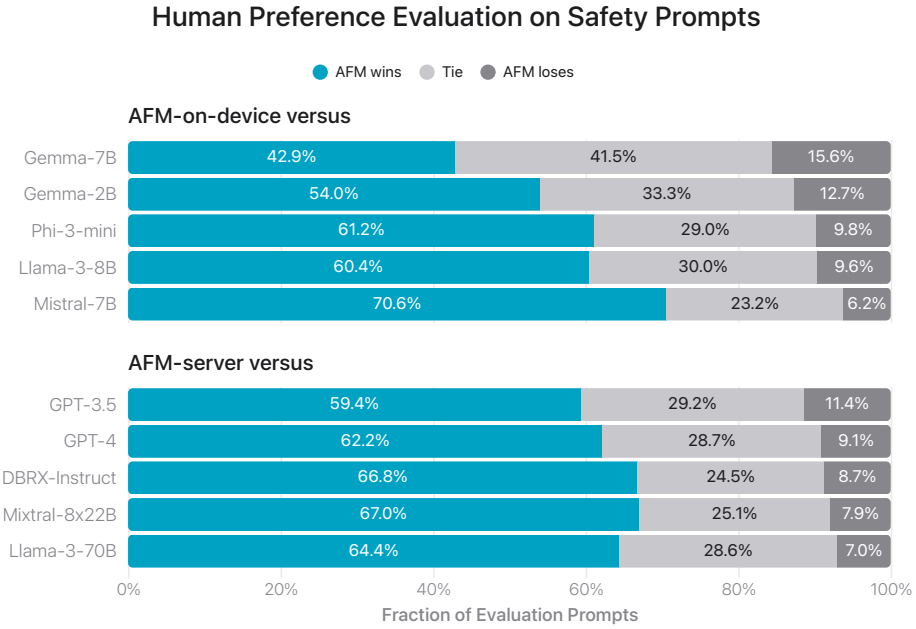

Figure 10: Fraction of preferred responses in side-by-side evaluation of Apple's foundation model against comparable models on safety prompts. Human graders found our responses safer and more helpful.

8 Conclusion

In this report we introduced the foundation language models that power Apple Intelligence features, AFM-on-device and AFM-server. The models are designed to be fast and run efficiently on iPhone, iPad, and Mac as well as on Apple silicon servers via Private Cloud Compute. They are trained to be highly capable in tasks like language understanding, instruction following,

reasoning, writing, and tool use. We have developed an innovative model architecture to specialize these models for our users' most common tasks. On top of the foundation models, feature-specific adapters are fine-tuned to provide high-quality user experiences such as summarization of emails, messages, and notifications. Our models have been created with the purpose of helping users do everyday activities across their Apple products, grounded in Apple's core values, and rooted in our Responsible AI principles at every stage. These foundation models are at the heart of Apple Intelligence, the personal intelligence system built by Apple to continue empowering our users and enriching their lives.

References

Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702. PMLR, 2019.

Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, pages 419–434, 2020.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in LLMs. 2024, arXiv:2402.14740.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023. doi: 10.18653/v1/2023.emnlp-main.298.

Apple. The AXLearn library for deep learning. https://github.com/apple/axlearn, 2023.

Apple. Applebot description. https://support.apple.com/en-us/HT204683, 2024a. Accessed: 2024-05-04.

Apple. Private cloud compute: A new frontier for ai privacy in the cloud. https://security.apple.com/blog/private-cloud-compute/, 2024b. Accessed: 2024-07-11.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International*

Conference on Artificial Intelligence and Statistics, pages 4447–4455. PMLR, 2024, arXiv:2310.12036.

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

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

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, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. 2022, arXiv:2204.02311.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024, arXiv:2210.11416.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. 2021, arXiv:2110.14168.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*, 36, 2024, arXiv:2305.14314.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. 2024, arXiv:2404.04475.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. 2022, arXiv:2210.17323.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. 2022, arXiv:2209.07858.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. 2021, arXiv:2103.03874.

Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 2015, arXiv:1503.02531.

Fred Hohman, Chaoqun Wang, Jinmook Lee, Jochen Görtler, Dominik Moritz, Jeffrey P Bigham, Zhile Ren, Cecile Foret, Qi Shan, and Xiaoyi Zhang. Talaria: Interactively optimizing machine learning models for efficient inference. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2024, arXiv:2404.03085.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? 2024, arXiv:2404.06654.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021, arXiv:2106.09685.

Huggingface. Open LLM evaluation. https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard, 2024. Accessed: 2024-07-09.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: LLM-based input-output safeguard for human-ai conversations. 2023, arXiv:2312.06674.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, page 441–450, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588896. doi: 10.1145/1718487.1718542.

Xiang Kong, Tom Gunter, and Ruoming Pang. Large language model-guided document selection. 2024, arXiv:2406.04638.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! 2019. URL https://openreview.net/forum?id=r1lgTGL5DE.

Nevena Lazic, Dong Yin, Yasin Abbasi-Yadkori, and Csaba Szepesvari. Improved regret bound and experience replay in regularized policy iteration. In *International Conference on Machine Learning*, pages 6032–6042. PMLR, 2021, arXiv:2102.12611.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36:8152–8172, 2023, arXiv:2304.04947.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35: 3843–3857, 2022.

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, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models. 2024a, arXiv:2406.11794.

Tianle* Li, Wei-Lin* Chiang, Evan Frick, Lisa Dunlap, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From live data to high-quality benchmarks: The arena-hard pipeline. April 2024b. URL https://lmsys.org/blog/2024-04-19-arena-hard/.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto,

Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. 2023, arXiv:2211.09110.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024, arXiv:2306.00978.

Xiaoran Liu, Hang Yan, Shuo Zhang, Chenxin An, Xipeng Qiu, and Dahua Lin. Scaling laws of rope-based extrapolation. 2024, arXiv:2310.05209.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019, arXiv:1711.05101.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018, arXiv:1712.01312. URL https://openreview.net/forum?id=H1Y8hhg0b.

Toan Q. Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. In Jan Niehues, Rolando Cattoni, Sebastian Stüker, Matteo Negri, Marco Turchi, Thanh-Le Ha, Elizabeth Salesky, Ramon Sanabria, Loic Barrault, Lucia Specia, and Marcello Federico, editors, *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong, November 2-3 2019. Association for Computational Linguistics. URL https://aclanthology.org/2019.iwslt-1.17.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022, arXiv:2203.02155.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. 2023, arXiv:2305.15334.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2016, arXiv:1608.05859.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024, arXiv:2305.18290.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015, arXiv:1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017, arXiv:1707.06347.

Noam Shazeer. Glu variants improve transformer, 2020, arXiv:2002.05202. URL https://arxiv.org/abs/2002.05202.

Stanford. HELM Lite: Lightweight and broad capabilities evaluation. https://crfm.stanford.edu/helm/lite/v1.5.0/, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

Manan Tomar, Lior Shani, Yonathan Efroni, and Mohammad Ghavamzadeh. Mirror descent policy optimization. 2020, arXiv:2005.09814.

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, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. 2023, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017, arXiv:1706.03762.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, 2020, arXiv:1910.04732. doi: 10.18653/v1/2020.emnlp-main.496.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. 2023, arXiv:2309.14322.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. 2023, arXiv:2310.06694.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. WizardLM: Empowering large language models to follow complex instructions. 2023, arXiv:2304.12244.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. 2022, arXiv:2203.03466.

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

Yi Zeng, Hongpeng Lin, Jingwen Zhang, Diyi Yang, Ruoxi Jia, and Weiyan Shi. How johnny can persuade llms to jailbreak them: Rethinking persuasion to challenge ai safety by humanizing llms, 2024, arXiv:2401.06373. URL https://arxiv.org/abs/2401.06373.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/1e8a19426224ca89e83cef47f1e7f53b-Paper.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-judge with MT-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. LIMA: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

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, arXiv:2311.07911.

Contributors

Within each section, contributors are listed in alphabetical order by first name.

Foundation Models

Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chong Wang (inference efficiency lead), Chung-Cheng Chiu, David Qiu, Deepak Gopinath, Dian Ang Yap, Dong Yin, Feng Nan, Floris Weers, Guoli Yin, Haoshuo Huang, Jianyu Wang, Jiarui Lu, John Peebles, Ke Ye, Mark Lee, Nan Du, Qibin Chen, Quentin Keunebroek, Ruoming Pang (overall lead), Sam Wiseman, Syd Evans, Tao Lei, Tom Gunter (pre-train lead), Vivek Rathod, Xiang Kong, Xianzhi Du, Yanghao Li, Yongqiang Wang, Yuan Gao, Zaid Ahmed, Zhaoyang Xu, Zhiyun Lu, Zirui Wang (post-train lead)

Data, Evaluation, and Responsible AI

Al Rashid, Albin Madappally Jose, Alec Doane, Alfredo Bencomo, Allison Vanderby, Andrew Hansen, Ankur Jain, Anupama Mann Anupama, Areeba Kamal, Bugu Wu, Carolina Brum, Charlie Maalouf, Chinguun Erdenebileg, Chris Dulhanty, Dominik Moritz, Doug Kang, Eduardo Jimenez, Evan Ladd, Fangping Shi, Felix Bai, Frank Chu, Fred Hohman, Hadas Kotek, Hannah Gillis Coleman, Jane Li, Jeffrey Bigham, Jeffery Cao, Jeff Lai, Jessica Cheung, Jiulong Shan, Joe Zhou, John Li, Jun Qin, Karanjeet Singh, Karla Vega, Ke Ye, Kelvin Zou, Laura Heckman, Lauren Gardiner, Margit Bowler, Mark Lee, Maria Cordell, Meng Cao, Nicole Hay, Nilesh Shahdadpuri, Otto Godwin, Pranay Dighe, Pushyami Rachapudi, Ramsey Tantawi, Roman Frigg, Sam Davarnia, Sanskruti Shah, Saptarshi Guha, Sasha Sirovica, Shen Ma, Shuang Ma, Simon Wang, Sulgi Kim, Suma Jayaram, Vaishaal Shankar, Varsha Paidi, Vivek Kumar, Xiang Kong, Xin Wang, Xin Zheng, Walker Cheng, Yael Shrager, Yang Ye, Yasu Tanaka, Yihao Guo, Yunsong Meng, Zhao Tang Luo, Zhi Ouyang, Zhiyun Lu

Adapters, Optimizations, and Summarization

Alp Aygar, Alvin Wan, Andrew Walkingshaw, Andy Narayanan, Antonie Lin, Arsalan Farooq, Brent Ramerth, Chong Wang, Colorado Reed, Chris Bartels, Chris Chaney, David Riazati, Eric Liang Yang, Erin Feldman, Gabriel Hochstrasser, Guillaume Seguin, Guoli Yin, Irina Belousova, Jianyu Wang, Joris Pelemans, Karen Yang, Keivan Alizadeh Vahid, Liangliang Cao, Mahyar Najibi , Marco Zuliani, Max Horton, Minsik Cho, Nikhil Bhendawade, Patrick Dong, Piotr Maj, Pulkit Agrawal, Qi Shan, Qibin Chen, Qichen Fu, Regan Poston, Sam Xu, Shuangning Liu, Sushma Rao, Tashweena Heeramun, Thomas Merth, Uday Rayala, Victor Cui, Vivek Rangarajan Sridhar, Vivek Rathod, Wencong Zhang, Wenqi Zhang, Wentao Wu, Xiang Kong, Xingyu Zhou, Xinwen Liu, Yang Zhao, Yin Xia, Zhile Ren, Zhongzheng Ren

Appendix

A Core pre-training recipe ablation

We compare our chosen settings for 'core' pre-training from Section 3.2.1 (optimizer, scaling-law-predicted batch-size, weight-decay, etc.) to a baseline based on [Wortsman et al., 2023]. In particular, the baseline uses AdamW with a standard hyperparameter configuration of $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 1e{-}15$, and a decoupled weight decay of 1e−4, decaying the learning rate to 0.0001 of peak, with a batch size of 1024 sequences. Otherwise both recipes are identical. Training covers 3.1T tokens using the AFM-on-device architecture but with a different data mixture to that used by the official AFM training runs.

Task	Baseline (acc)	AFM (acc)
arc_challenge	41.9	44.6
arc_easy	75.6	76.1
hellaswag	54.3	55.0
lambada	69.3	68.9
piqa	78.3	78.4
sciq	94.5	94.7
winogrande	67.3	66.9
triviaqa (1-shot)	40.5	41.0
webqs (1-shot)	20.6	20.6
CoreEN average	60.2	60.7
GSM8K (8-shot CoT)	16.6	18.9
MMLU (5-shot)	45.4	45.5

Table 5: Core pre-training recipe ablation few-shot results. Unless otherwise noted, we use 0-shot prompts. We note that AFM's recipe allows for slight improvements across the majority of tasks, although the difference is typically very small. The data mixture differs from the official AFM runs.

In Table 5, AFM's recipe demonstrates a slight improvement over the baseline. This likely indicates that the most important recipe settings are already well-enough configured by the baseline for this model size and training budget.

B Ablations on pruning and distillation

Here we detail the evaluation results of using structural pruning and distillation separately and show they can be combined together to get the best performance.

Table 6 shows the ablation results of training 3B models using an early version of our pre-training data mixture. As shown in the table, both pruning

and distillation methods can outperform a baseline model trained from scratch. For example, pruning and distillation achieve a MMLU score of 42.9% and 44.9% respectively, whereas a baseline using 50% more steps gets 34.6%. It is also interesting that pruning achieves a higher score on the CoreEn benchmark, while distillation is better on MMLU. Finally, when combining these two methods together, we observe further improvements on MMLU and GSM8k by a large margin, getting better or on par results compared to the baseline trained using 5× more computation.

Metric/Method	Baseline	Prune	Distill	Both	Baseline
Training cost	1.5×	1×	1×	1×	5×
MMLU (5-shot)	34.6	42.9	44.9	**49.3**	45.4
GSM8K (8-shot CoT)	12.7	13.5	11.0	**16.8**	**16.9**
CoreEN Average	59.8	**61.0**	58.1	59.7	60.3

Table 6: Ablation results of pruning and distillation methods. The training data is an early version that differs from the official AFM runs.

C Pre-training stage-breakdown evaluations

We present few-shot evaluation results after core, continued, and long-context pre-training stages, for a subset of evaluation metrics that we find to be low-variance, diverse, and correlated with downstream evaluation after post-training. These metrics are derived using an internal harness and set of benchmark formulations, which are not optimized for absolute performance (e.g. we do not apply length normalization, and use more difficult test splits where available—for TriviaQA as one example). They are therefore not suitable for comparison with other published results.

In Table 7 and 8 we present internal benchmarks after all three stages of pre-training. As expected, continued pre-training acts to improve math and particularly code model capabilities, whilst subtly improving a few other benchmarks. The context-lengthening stage leaves the majority of these benchmarks on-par, with changes (positive and negative) typically within the range of what we consider to be evaluation noise.

D Long-context evaluation

Although the focus for this version of AFM was not to support context lengths longer than 8k, in Table 9 we use the RULER [Hsieh et al., 2024] benchmark to evaluate AFM-server at 4k to 32k context lengths. We note that the model is capable of performing perfectly up to a sequence length of $\geq$ 32k when tested against straightforward retrieval-like tasks, e.g., needle-in-the-haystack (NIAH). It is clear, however, that the model performance gradually suffers with an

AFM-on-device	Core	Continued	Context lengthened
ARC_C	43.17	47.53	45.39
ARC_E	74.87	78.62	78.37
HellaSwag	54.70	55.50	55.24
LAMBADA	73.51	70.13	69.90
PIQA	77.37	78.67	78.40
SciQ	94.90	95.80	95.70
WinoGrande	65.82	67.32	67.01
TriviaQA (1 shot)	42.46	39.13	38.11
WebQS (1 shot)	19.24	18.06	17.22
CoreEN average	60.67	61.20	60.59
MMLU (5 shot)	57.00	61.35	60.64
GSM8K (8 shot CoT)	27.45	42.53	40.00
MATH (4 shot CoT)	8.31	16.97	15.48
HumanEval-Py pass@1	16.48	27.38	30.84
MultiPLE-Swift pass@1	8.88	19.24	18.06

Table 7: Pre-training evaluation for AFM-on-device with an internal harness. Unless otherwise noted, we use 0-shot prompts. TriviaQA evaluation is on the larger and more challenging "Web" split.

increasing context length on RULER, a more complex evaluation benchmark than NIAH, suggesting that the real context length for AFM-server, for tasks beyond retrieval, is currently at most 24k.

E Technical details for RLHF

E.1 Reward modeling

The human preference data that we use in reward model training has the following format:

- x: the prompt;
- y_c: the chosen (preferred) response;
- y_r: the rejected response;
- ℓ: the level of the human preference;
- z_c^{if} and z_r^{if}: the instruction-following property of the two responses;
- z_c^{verb} and z_r^{verb}: the verbosity of the two responses;
- z_c^{truth} and z_r^{truth}: the truthfulness of the two responses;

AFM-server	Core	Continued	Context lengthened
ARC_C	58.28	58.87	57.94
ARC_E	85.61	85.44	85.06
HellaSwag	64.17	64.53	64.37
LAMBADA	78.38	77.59	77.82
PIQA	82.37	81.99	81.88
SciQ	96.60	97.10	97.00
WinoGrande	80.51	79.16	79.08
TriviaQA (1 shot)	54.33	53.57	53.42
WebQS (1 shot)	29.97	27.66	27.41
CoreEN average	70.02	69.55	69.33
MMLU (5 shot)	74.00	75.24	74.80
GSM8K (8 shot CoT)	75.44	74.83	75.51
MATH (4 shot CoT)	32.24	36.48	35.77
HumanEval-Py	33.23	40.77	39.55
MultiPLE-Swift	30.15	37.70	38.11

Table 8: Pre-training evaluation for AFM-server with an internal harness. Unless otherwise noted, we use 0-shot prompts. TriviaQA evaluation is on the larger and more challenging "Web" split.

AFM-server	Average acc
Ctx @ 4096	91.7
Ctx @ 8192	87.7
Ctx @ 16384	84.1
Ctx @ 20480	79.1
Ctx @ 24576	75.8
Ctx @ 32768	43.3

Table 9: RULER [Hsieh et al., 2024] average evaluation results, averaged over 13 synthetic long-context tasks using 500 examples per task.

- z_c^{harm} and z_r^{harm}: the harmlessness of the two responses.

In our reward modeling, the preference level ℓ takes 4 possible values, indicating that the chosen response is negligibly better, slightly better, better, or significantly better than the rejected response. As for the single sided gradings, each label, e.g., z_c^{if}, takes 3 possible values. For instruction following, truthfulness, and harmlessness, the 3 values correspond to the cases where the response has major issue, minor issue, or no issue. For verbosity, the 3 values

correspond to the cases where the response is too verbose, too short, or just right.

We use a multi-head architecture for the reward model. More specifically, we take a decoder-only transformer and obtain the last-layer embedding of the last non-padding token. We attach one linear and four MLP heads to the embedding. Denote the model parameters by ϕ and the input prompt-response pair by (x, y). The linear head outputs the preference reward $r_\phi(x, y) \in \mathbb{R}$. The four MLP heads are classification heads representing the instruction-following, verbosity, truthfulness, and harmlessness property of the response. We denote the output logits of the 4 classification heads by u_ϕ^{if}, u_ϕ^{verb}, u_ϕ^{truth}, u_ϕ^{harm}, respectively.

Soft label loss. We train the preference reward $r_\phi(x, y)$ based on Bradley-Terry-Luce (BTL) model [Bradley and Terry, 1952]. Recall that in BTL model, the probability that y_c is preferred over y_r is modeled as $\sigma(r_\phi(x, y_c) - r_\phi(x, y_r))$, where σ is the sigmoid function. Intuitively, this probability should be larger if the preferred response y_c is annotated as significantly better than the rejected response y_r, and smaller if y_c is only negligibly better than y_r. We incorporate this information using the preference level ℓ. More specifically, for each preference level ℓ, we design a *target* preference probability p_ℓ. Then we use a *soft label* loss as follows:

$$\begin{aligned} L_{\text{ranking}}(\phi) = &- p_\ell \log(\sigma(r_\phi(x, y_c) - r_\phi(x, y_r)) \\ &- (1 - p_\ell) \log(\sigma(r_\phi(x, y_r) - r_\phi(x, y_c)). \end{aligned} \tag{3}$$

The target level p_ℓ is a hyperparameter in our algorithm and should take larger value if the preference level is higher. In our experiments, we choose $p_\ell = 0.95,\ 0.85,\ 0.75,\ 0.65$ for significantly better, better, slightly better, and negligibly better, respectively.

Single-sided grading as regularization. We also leverage the single-sided gradings as regularization terms in our reward model. The intuition is that with these gradings as regularization terms, we can learn a better embedding to capture human preferences. The regularization loss is

$$\begin{aligned} L_{\text{regu}}(\phi) = \sum_{\text{grade} \in \text{if,verb,truth,harm}} \Big(&\mathsf{cross\_entropy}(u_\phi^{\text{grade}}(x, y_c), z_c^{\text{grade}}) \\ &+ \mathsf{cross\_entropy}(u_\phi^{\text{grade}}(x, y_r), z_r^{\text{grade}}) \Big). \end{aligned} \tag{4}$$

Overall, the reward model training loss that we use is

$$L_{\text{ranking}}(\phi) + \lambda L_{\text{regu}}(\phi). \tag{5}$$

E.2 Online RL algorithm

In this section, we present more details of our online RLHF algorithm, MDLOO.

Leave-One-Out (LOO) estimator of the advantage. In each iteration of the algorithm, we have a data collection stage and a policy updating stage. Let θ_k be the model parameter at the beginning of the k-th iteration. We sample a batch of n prompts from our prompt set, and for each prompt, we sample K responses according to the policy π_{θ_k}, and thus collecting a total of nK data points in each iteration. Let x be a prompt and y_i be one of the responses. Since we consider the bandit setting, by definition, the *advantage* of (x, y_i) is

$$A_k(x, y_i) = R(x, y_i) - \mathbb{E}_{y\sim\pi_{\theta_k}(\cdot|x)}[R(x, y)]. \tag{6}$$

We use the leave-one-out (LOO) method [Kool et al., 2019] to estimate $A_k(x, y_i)$. Namely, we estimate the mean reward given the prompt x with the other $K-1$ responses, i.e.,

$$\widehat{A}_k(x, y_i) = R(x, y_i) - \frac{1}{K-1}\sum_{j\neq i} R(x, y_j). \tag{7}$$

As shown in recent works [Ahmadian et al., 2024], this advantage estimation is beneficial for RLHF. Empirically, we find that using LOO estimator leads to more stable training and better results compared to directly using the reward as the advantage estimation or using the difference between the reward and a running average baseline [Williams, 1992].

Mirror descent policy optimization (MDPO). Our policy optimization approach belongs to a widely used class of trust-region policy optimization algorithms [Schulman et al., 2015]. The basic idea in these algorithms is that in each policy iteration, we apply a regularization method to prevent the policy from changing too much in an iteration. The regularization can be achieved by adding KL regularization [Abbasi-Yadkori et al., 2019; Lazic et al., 2021; Tomar et al., 2020] and using clipping for the probability ratio such as in PPO [Schulman et al., 2017]. In this work, we use KL regularization as in Mirror Descent Policy Optimization (MDPO) [Tomar et al., 2020].

In particular, in the k-th iteration, with the data (prompts along with the K responses sampled according to π_{θ_k} for each prompt), we aim to optimize the following regularized advantage maximization problem:

$$\max_{\theta} \Psi(\theta) := \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{y\sim\pi_{\theta_k}(\cdot|x)}[A_k(x, y)] - \gamma D_{\mathrm{KL}}(\pi_\theta(\cdot|x)||\pi_{\theta_k}(\cdot|x))\right]. \tag{8}$$

Note that here the KL regularization term is different from the one in Eq. (1). The KL regularization in Eq. (1) is between the policy model and the reference model; whereas the KL regularization term in Eq. (8) is between the policy model and the policy at the beginning of the k-th iteration. Then we can obtain the gradient of $\Psi(\theta)$ as

$$\begin{aligned}\nabla\Psi(\theta) =& \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_{\theta_k}(\cdot|x)}\left[\frac{\pi_\theta(y|x)}{\pi_{\theta_k}(y|x)} A_k(x, y)\nabla\log\pi_\theta(y|x)\right] \\ &- \gamma\mathbb{E}_{x\sim\mathcal{D}}\left[\nabla D_{\mathrm{KL}}\pi_\theta(\cdot|x)||\pi_{\theta_k}(\cdot|x)\right].\end{aligned} \tag{9}$$

The MDLOO algorithm can be derived by replacing the expectations in Eq. (9) with the nK samples collected with π_{θ_k}, and the advantage $A_k(x, y)$ with the LOO estimator $\widehat{A}_k(x, y)$ in Eq. (7). Empirically, we find that MDLOO works better than the popular PPO [Schulman et al., 2017] algorithm in our setting.

F Accuracy-recovery adapters ablation

In this section, we present the evaluation results on unquantized, quantized, and accuracy-recovered models. As shown in Table 10, the quantized models have huge quality drops in both pre-train and post-train metrics. By using accuracy-recovery LoRA adapters with only rank 16, Alpaca win rate can be improved by 7-18%, GMS8K accuracy is boosted by 5-10%. The recovered models perform much closer to the original unquantized model while achieving significant reductions on the model size. More interestingly, we observe that when the quantization scheme becomes more aggressive (from 3.7 to 3.5 bpw), the adapters also recover more quality back.

BPW	Models	IFEval Instruction-Level	AlpacaEval 2.0 LC	GSM8K (8-shot CoT)
16	**AFM-on-device**	100.0%	100.0%	100.0%
3.5	**quantized**	98.4%	76.7%	82.2%
	Acc.-recovered (rank 16)	98.8%	94.7%	92.1%
3.7	**quantized**	97.9%	87.3%	91.3%
	Acc.-recovered (rank 16)	100.6%	94.8%	96.0%

Table 10: Evaluation results for quantized and accuracy-recovered models. Numbers are normalized to the unquantized version.