360Brew : A Decoder-only Foundation Model for Personaliz Ranking and Recommendation

360Brew Team
Foundation AI Technologies (FAIT), LinkedIn
Corresponding Authors: Maziar Sanjabi*, Hamed Firooz†

Abstract

Ranking and recommendation systems constitute the foundation for numerous online experiences, ranging from search results to personalized content delivery. These systems have evolved into complex, multilayered architectures that leverage vast datasets and often incorporate thousands of predictive models. The maintenance and enhancement of these models is a labor intensive process that requires extensive feature engineering. This approach not only exacerbates technical debt but also hampers innovation in extending these systems to emerging problem domains.

In this report, we present our research aimed at addressing these challenges by leveraging a large foundation model with a textual interface for ranking and recommendation tasks the LinkedIn platform[1]. We highlight several key advantages of our approach in contrast to mainstream methods, which are predominantly ID-based. Specifically, our approach: (1) a single model can manage multiple predictive tasks involved in ranking and recommendation within LinkedIn's services, (2) decoder models with textual interface due to their comprehension of reasoning capabilities, can generalize to new recommendation surfaces and out-of-domain ranking and retrieval tasks in a zero-shot manner through simple prompts, thereby enabling product designers to engage with and iterate on them with ease, and (3) by employing natural language interfaces for task definitions and verbalizing member behaviors and their social connections, we replace traditional cumbersome feature engineering and complex directed acyclic graphs (DAGs) of model dependencies with centralized prompt engineering. We introduce our research on our pre-production model, *360Brew* V1.0, developed by a small team of researchers and engineers over a 9-month period. *360Brew* is a 150B parameter, decoder-only model that has been trained and fine-tuned on LinkedIn's primarily first-party data and tasks (from users outside European Union). This model is capable of solving over 30 predictive tasks across various segments of the LinkedIn platform, achieving performance levels comparable to or exceeding those of current production systems based on offline metrics, without task-specific fine-tuning. Notably, each of these tasks is conventionally addressed by dedicated models that have been developed and maintained over multiple years by teams of a similar or larger size than our own.

The primary objective of *360Brew* is to enhance AI development productivity across LinkedIn by centralizing common modeling components into a foundational layer. This approach facilitates fundamental modifications, moderation, and governance at the foundational level, ensuring simultaneous benefits across all platforms. Additionally, developers are empowered to rapidly iterate on new objectives, data sources, and AI-driven search and recommendation products.

1 Introduction

Production-level Recommendation Systems (RS) are built to provide a better experience for Internet users by matching them with the content such as posts, jobs, people. These systems are at the heart of many internet applications, e.g., search, social networking, e-commerce, and many more [1, 2]. RS often consists of many layers and models. In Figure 1 we show a simplified version of such complex systems. On the one side are generally members whom the systems want to recommend jobs, posts,

*maz@linkedin.com
†hfirooz@linkedin.com

[1]Please note that the language models referred to in this paper are only used for ranking and recommender systems on LinkedIn, and are not used for content creation features on LinkedIn.

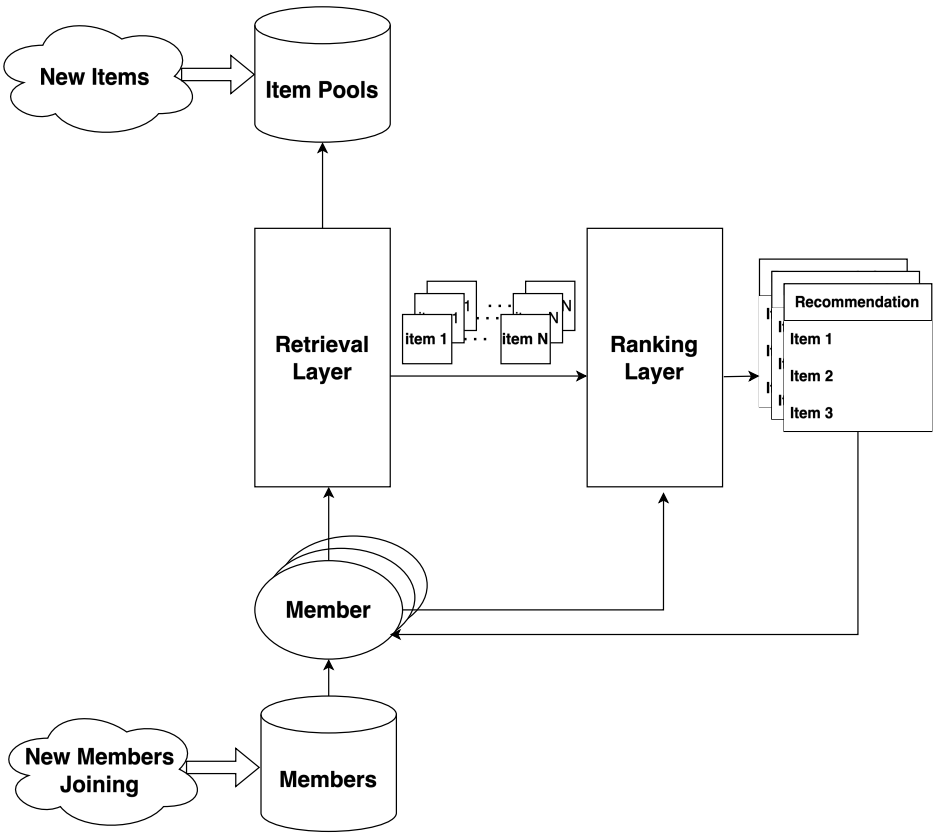

Figure 1: Overview of A Recommendation System (RS)

or people to connect with. The load on the system is defined in terms of queries per second, which in pull-based recommendations is dictated by the member count and their frequency of interactions. For example, if it is a search system, it is decided by how often people search on the platform. On the other side of the recommendation, members often need to be matched with items, e.g. jobs or posts. In most cases, the size of the item pool is much larger or at least equal to the size of the member pool. For example, the number of posts on LinkedIn is much larger than the number of LinkedIn members. Due to the large size of the item pool, the problem often needs to be solved in multiple stages, depicted here as retrieval and ranking layers[2]. In the retrieval layer, the idea is to quickly narrow down the number of possible items per member. In the ranking layer, the goal is to find a few best items among the retrieved items and rank them for the member to consume.

While both retrieval and ranking are trying to fundamentally solve the same problem (find the N best items for the member in a pool of K items), they have their own goals and challenges. For example, since retrieval is at the beginning of the RS funnel (large K and large N), it needs to have high recall. Moreover, due to large K, retrieval needs to be super-scalable and often uses simple member-item interaction architectures such as two tower models [3]. On the other hand, ranking happens after the retrieval and has a smaller number of potential items to filter (smaller K). This allows the usage of more complicated models, e.g. DCN [4]. At the same time, since the ranking model output is member-facing, it needs to have high precision (since N is small) and high recall to result in a good member experience.

In this work on our *360Brew* model V1.0, we mainly focus on ranking and re-ranking tasks since they need to have high precision and recall and are less bounded by computational constraints in practice (as mentioned above). We believe that it is possible to have the same model for retrieval (through embeddings, e.g. [5]) and ranking (through predictions), and it would be the focus of our next *360Brew* model releases.

[2]The retrieval and ranking layers themselves can possibly consist of multiple sub-stages.

1.1 Status Quo of RS Models

The retrieval and ranking layers in RS typically rely on ID based features for members and items. Essentially, these are large embedding tables where each embedding represents a (few) member(s)/item(s). In addition to the ID based features, manually engineered features such as member/item interactions and item attributes are employed for richer representation learning. However, this process demands a perpetual process of improving and maintaining these models by large engineering teams.

In the rest of this section we discuss the challenges and active areas of research in RS modeling. Interestingly these challenges are mostly a direct result of using ID based and hand-crafted features.

- **Cold-start**: The RS models rely on currently available IDs and cannot generalize to new members and items without frequent re-training. This frequent and computationally heavy process is necessary for adapting to new items/members and varying member behaviors (see the following points). As a result, a large body of works have been dedicated to solving this issue through approaches such as content-based representations [6, 7].
- **Interactions**: Older RS models were primitive and required most of the cross features between member(s) and item(s) to be hand-crafted. With the popularity of deep neural networks (DNNs), architectures, such as DCN [4], which would allow automatic crossing of the item and member features became popular. More recently, there have been many works proposing more advanced architectures, e.g. transformers, for this purpose [8, 9].
- **Domain generalization**: Most of the RS models are super-specialized, through the use of ID based and specialized embedding features. Such models are trained on specific interactions from a specific domain and cannot generalize to new surfaces and tasks [10]. For example, they cannot go from feed recommendation to job recommendation (surface change), or start predicting member's interaction instead of member's ratings (task change).

In this work we introduce *360Brew* which is LinkedIn's venture project to build RS models based on modern and powerful LLMs that can naturally address the above-mentioned drawbacks of traditional RS models.

2 *360Brew*

2.1 Overview

Foundation models have achieved an unprecedented level of generality by leveraging scale and attention-based architectures [11, 12]. Recent advancements in decoder-only model architectures that use language as an input interface have demonstrated their capabilities in understanding, reasoning, and solving a variety of tasks beyond natural language processing, including graph understanding [13, 14, 15], robotic planning [16], and knowledge base management [17, 18], among others.

Recent research has explored the application of large pre-trained decoder-only models to member-item recommendation tasks [19, 10]. The emergent problem-solving and reasoning capabilities of these models allow them to process member and item information—such as member profiles and job or post descriptions—and assess their relevance with high accuracy. **This capability enables the models to exhibit improved performance on cold-start problems without additional modifications**.

Traditional ranking models often rely on hundreds of thousands of handcrafted features and complex architectures to capture high-order interactions among these features [4]. In contrast, the deep, multi-layer transformer architecture of LLMs can extract the necessary features directly from contextual information that is provided in text to address the task at hand [20]. **This approach centralizes feature engineering for items or members into prompt engineering, since all input is processed as text by the model**. Furthermore, LLMs are **adept at generalizing to different item types and interactions, enhancing their out-of-the-box ability to adapt to new items, surfaces and interaction patterns**. This property supports broad generalizability in new contexts.

In many practical use cases, traditional RS utilize text and content understanding models solely as feature generators to enhance ID-based models [6, 7]. However, our approach goes a step further. Instead of merely using these models to produce embeddings or features for the system, we aim to replace the entire model with an LLM that employs a natural language interface. **This approach**

is motivated by the well-documented ability of such models to identify and generalize patterns, as demonstrated by their reasoning capabilities [21] and few-shot problem-solving performance [22]. As we discuss in the following sections, this capability is vital for highly fine-grained personalization in our RS.

In any recommendation task, each member, with their unique profile and history of interactions, can be viewed as a many-shot problem [23]. Consequently, when the ranking model is conditioned on the member's profile and interaction history, it can identify and generalize patterns that are highly personalized for that member, extending these patterns to future interactions. With this setup, we can reformulate a recommendation problem as solving tasks through **many-shot in-context learning (ICL)**. In other words, our foundation model is trained to approximate the following joint distribution for all members, interactions, and tasks:

$$P(\ m,\ (e_1,\ i_1), \cdots,\ (e_{T-1},\ i_{T-1}),\ (e_T,\ i_T)), \tag{1}$$

where m denotes the textual member profile, and each pair (e_t, i_t) for $t = 1, \ldots, T$ represents a historical entity e_t (such as a "post") together with the associated member interactions i_t (such as member "like" and "comment" on the post), both encoded in text. Note that i_t may contain multiple interactions, separated by commas.

Given this approximation of joint probability, the model can be used to estimate various marginal probabilities for different tasks:

$$P(i_t,\ i_{t+1}, \cdots |\ \text{Task Instruction},\ m,\ (e_1,\ i_1), \cdots,\ (e_{t-1},\ i_{t-1}),\ e_t,\ e_{t+1}, \cdots), \tag{2}$$

where "Task Instruction" represents the task, surface, or interaction description, and $e_t,\ e_{t+1}, \cdots$ are *the new entities for which we want to predict the member's interactions*. An example of a possible prompt is provided in Table 1. Note that this formulation is similar to the next token prediction approach used in training decoder-only LLMs. As a result, we can adapt this architecture for our foundation ranking model, which we call *360Brew* .

As previously mentioned, by starting our modeling process with a publicly available open-sourced pre-trained LLM architecture, we benefit from the model's inherent ability to understand items and members as separate entities. However, the model does not initially capture the relationships between them. By further training the model on these relationships, we enhance its few-shot capabilities, enabling it to identify complex patterns and extrapolate these to predict future interactions. This is achievable due to the architecture of large multi-layer transformers, which are adept at detecting and reasoning about intricate patterns [21, 22]. We will demonstrate this capability in the results section by showcasing the model's performance in solving a variety of in-domain and out-of-domain ranking and prediction tasks. These results are competitive with, or even surpass, those of current production models.

Traditional large-scale industrial RS rely heavily on tens of thousands of handcrafted features, including features designed for different time horizons, such as multi-resolution count features [24]. Recent advancements in DNN architectures have sought to reduce the dependency on engineered features by enabling the models to discover these features from large amounts of data [4, 25]. **Since the *360Brew* model relies solely on interaction history, it eliminates the need for hand-crafted, time-horizon-specific features, such as interactions per week, which are customary in traditional RS**.

As we will show in our results, an interesting byproduct of this approach is that more historical context improves the performance of the model. **This implies that we have replaced the problem of feature engineering with the more tractable challenge of scaling up architecture depth and context length, which primarily requires additional compute resources**. This highlights the advantage of LLM foundation models in optimizing human resource utilization by replacing the labor-intensive (and non-scalable) process of feature engineering with a compute-driven approach that relies on capital expenditure and is inherently scalable.

2.2 Results

In this section, we present the results from developing the *360Brew* V1.0 model. The *360Brew* model V1.0 is built on top of Mixtral 8x22 pre-trained MoE [26] architecture[3]. We further train it using

[3]

Instruction:
You are provided a member's profile and a set of jobs, their description, and interactions that the member had with the jobs. For each past job, the member has taken one of the following actions: applied, viewed, dismissed, or did not interact.
Your task is to analyze the job interaction data along with the member's profile to predict whether the member will apply, view, or dismiss a new job referred to as the "Question" job.

Note: Focus on skills, location, and years of experience more than other criteria. In your calculation, assign a 30% weight to the relevance between the member's profile and the job description, and a 70% weight to the member's historical activity.

Member Profile:
Current position: software engineer, current company: Google, Location: Sunnyvale, California.

Past job interaction data:
Member has applied to the following jobs: [Title: Software Engineer, Location: New York, Country: USA, Company: Meta, Description: ...]
Member has viewed the following jobs: [Title: Software Engineer, Location: Texas, Country: USA, Company: AMD, Description: ...]

Question:
Will the member apply to the following job: [Title: Software Engineer, Location: Seattle, Country: USA, Company: Apple, Description: ...]

Answer:
The member will apply

Table 1: A toy example of *360Brew* input for solving a job recommendation task. Note that the member and job data used in this example are entirely fictional.

a combination of LinkedIn raw entity data—such as member profiles, job descriptions, and LinkedIn posts—and interaction data (including member job applications) across 5+ surfaces on LinkedIn.

Finally, we apply the model to 30+ tasks across 8+ surfaces. Notably, not all tasks are in-domain, as some surfaces and tasks are outside the training data that the model has seen. Specifically, we categorize the tasks as follows:

- **T1** (in-domain): Tasks where recommendation data from past periods is used in training the model. There is at least a one-month gap between the T1 data used in training and the benchmark dataset. This means the data is subject to distribution shift, which is very common in recommendation systems, but the model has seen member behavior in the past for these tasks.
- **T2** (out-of-domain): Tasks and surfaces that are not part of the training data. In these tasks, the model is evaluated on data from domains or recommendation use cases that it has not encountered during training, representing an out-of-domain distribution.

Our model is flexible and can predict different type of outputs based on the prompt. However, for simplicity and ease of scaling, the majority of traditional large industry ranking models are framed as binary tasks, such as $P(\text{like})$. Therefore, we also adapt our prompts so that the model generates binary predictions when answering questions about whether a member likes a post, allowing for direct comparison with these models. We use the logits for these tokens to obtain scores and compute metrics such as AUC, etc.

2.3 Data Scaling

As mentioned earlier, one of the most compelling benefits of building foundation models for ranking is the reduced development time. The way that this goal is achieved is by training a single large model

on a large amount of data [27, 11]. In Figure 2, we show that as we scaled our processed data by over 3x[4], the model's performance on T1 tasks improved significantly compared to the baselines. Notably, the baseline models were developed through multi-year investments, each requiring dedicated teams of engineers for maintenance and ongoing development. In contrast, these data scalings were achieved within a nine-month period by a relatively small team. Furthermore, it is important to highlight that the slope of improvement with increased data size does not appear to be flattening. This indicates that our model continues to identify useful patterns as more data becomes available to it. As such, we anticipate further improvements as the model can access additional data in the future.

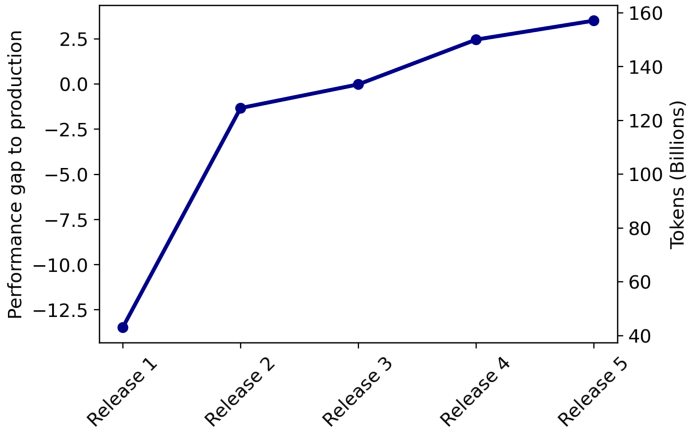

Figure 2: Effects of data size (in tokens) on the average performance of the model compared to baselines across five releases over a 9-month period.

2.4 Model Scaling

Recent works have shown increasing the size of the LLMs improves their capabilities in understanding the context and performing tasks [11]. In Figure 3 we show that the same holds in RS use-cases and *360Brew* models get better as we increase the size of the model (by using larger and more powerful pre-trained architectures). This shows that there is an added value in modeling more complex relationships or reasoning by using larger model architectures in RS use-cases.

2.5 History Scaling

As mentioned above, one advantage of *360Brew* model is that it does not require manual feature engineering. In Figure 4, we show that *360Brew* model's performance gets better as we increase the history by increasing the max context length. The take-home message here is that improving processed history (by using more compute) could be an effective way of improving the performance of the RS. It is worth noting that the story here is a bit more nuanced than other scaling laws since the performance of the model could be affected by the underlying model's limitation on its context length. For more details see We believe that the technical and modeling limitations on the context length will be alleviated with more advances in pre-trained LLM architectures. Based on our results, such advancements could directly benefit the *360Brew* model.

2.6 Generalization

2.6.1 Out of domain

In Figure 5 we show that the *360Brew* model can generalize to out-of-domain tasks and surfaces, and achieves performance similar to or better than the production model.

[4]

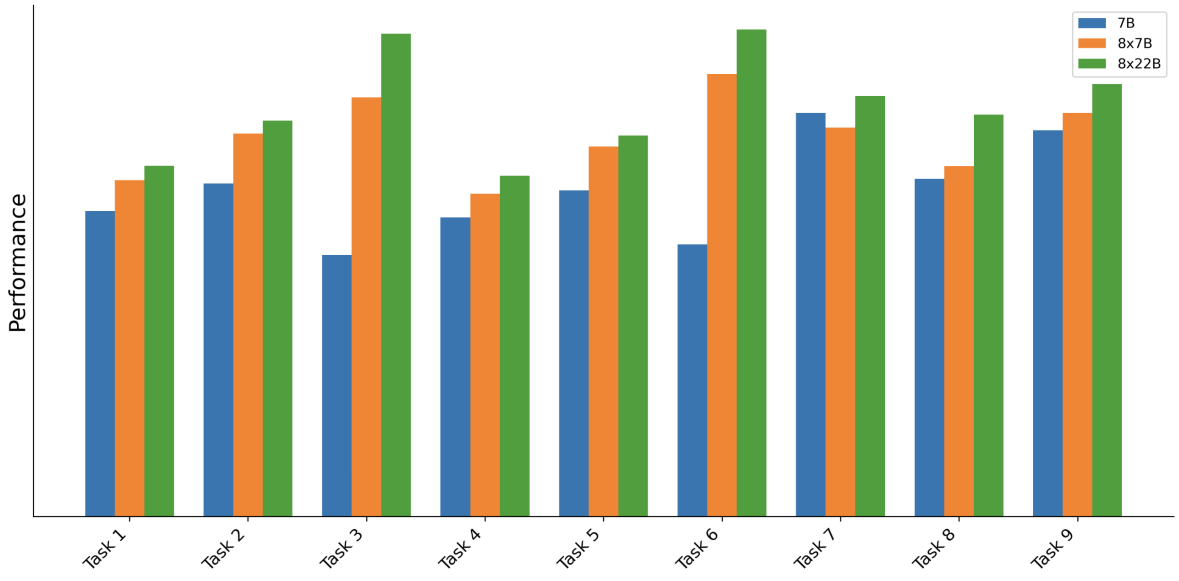

Figure 3: Effects of model size (in parameters) on the performance across different tasks.

2.6.2 Cold-start

In Figure 6 we show the gap between the *360Brew* model and the production model as max number of historical data points decreases from 100 to 5. As shown in the plot, the performance gap between the two models is the largest when member has few available interactions, which shows that *360Brew* model has a greater margin over the production model for members with fewer interactions. These are usually called cold start members since they do not have many interactions.

2.6.3 Temporal

One major downside of the current recommendation systems is that they do not generalize well over time, and the relationships that they have extracted might be irrelevant to the data distribution shifts. This results in frequent updating of the model which increases the maintenance cost, development time, and complexity of the systems. In Figure 7, we benchmark the performance of the *360Brew* model on different test data that lie within different time frames (in the future) from the training data. We do the same with the baseline models and show that the performance of the *360Brew* model is less affected by time. This means that using *360Brew* could potentially lead to more developer efficiency and less maintenance and technical debt as we do not need to update the model so frequently. This is possible since *360Brew* model can use ICL to adjust its answers based on the member behavior it sees in its context.

3 Infrastructure Scaling

3.1 Distributed Training

3.1.1 Training Loop

We leverage the Lightning Trainer [28] to set up our training framework, streamlining the training process and reducing boilerplate code associated with deep learning models. All training stages described in subsection 4.1 share the same Lightning Module but utilize different Lightning DataLoader modules tailored to their specific data requirements. This approach promotes code reusability and ensures consistency across various stages of training.

To efficiently utilize computational resources, we employ the Lightning Fully Sharded Data Parallel (FSDP) strategy for data parallelism. As described in detail in next section, FSDP enables us to distribute model parameters and optimizer states across multiple GPUs, allowing for the training of

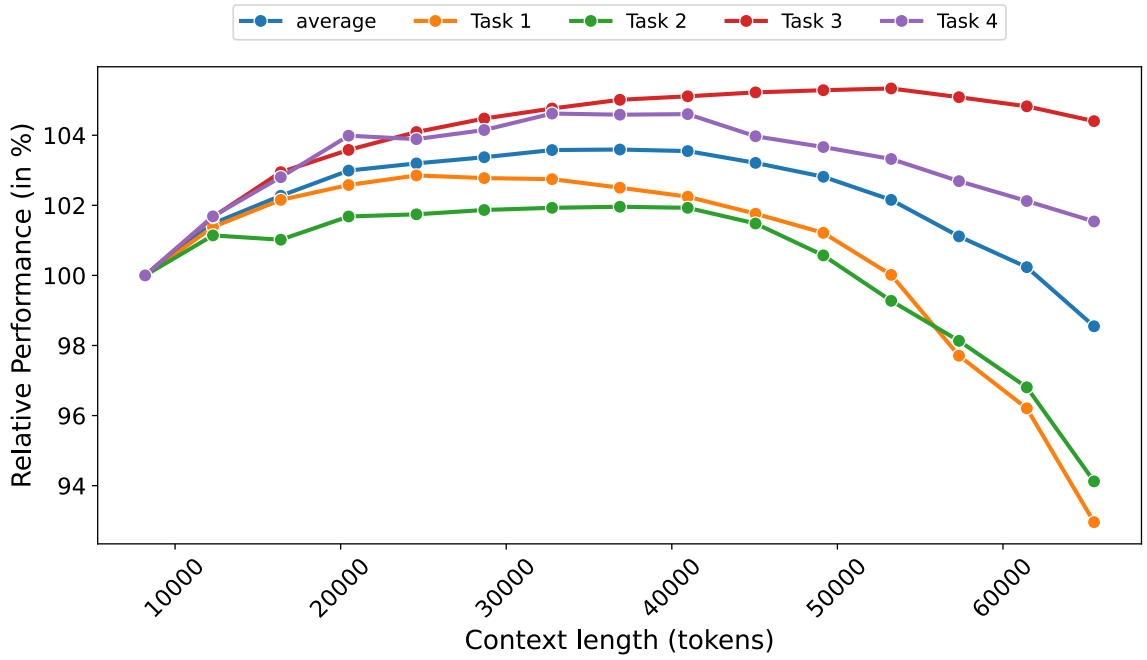

Figure 4: Effects of max context length (in tokens) on the performance of *360Brew* . The performance at each maximum context length is normalized by the performance at 8K maximum context length.

larger models that exceed the memory capacity of a single device. Additionally, we use Lightning's logging capabilities to record various observability metrics during training, such as loss values and resource utilization. This detailed logging facilitates monitoring the training process and aids in debugging and optimizing model performance.

3.1.2 Checkpoint and Restore

We developed a customized checkpoint callback that efficiently stores model weights, optimizer states, and data loader statuses in a distributed manner based on PyTorch Lightning APIs. The checkpointing process completes within minutes to minimize training disruption. This checkpoint enables complete job restoration, ensuring consistent model and data states. Additionally, it supports fault tolerance, automatically recovering from the latest checkpoint without human intervention.

3.1.3 Mixed-Precision

We leverage the mixed precision support in FSDP [29], and employ half-precision (bf16) for the model parameters, and use full precision (fp32) for model parameter updates and optimizer states. The full precision for the updates is crucial since these values accumulate over time and lower precisions hurt the performance significantly. Note that using lower precision for these values could have resulted in lower memory utilization and thus higher throughput in training. But in our use-case the higher throughput could not justify the drop in performance. Finally, it is worth noting that we did not experiment with approaches that keep the optimizer states at full precision but share optimizer states such as Adam-mini [30]

3.1.4 3D Parallelism

In order to scale effectively, we evaluated traditional "3D" parallelism (tensor-, pipeline-, and data-parallelism) as offered by frameworks like FairScale [31] and Megatron-LM [32]. We also considered

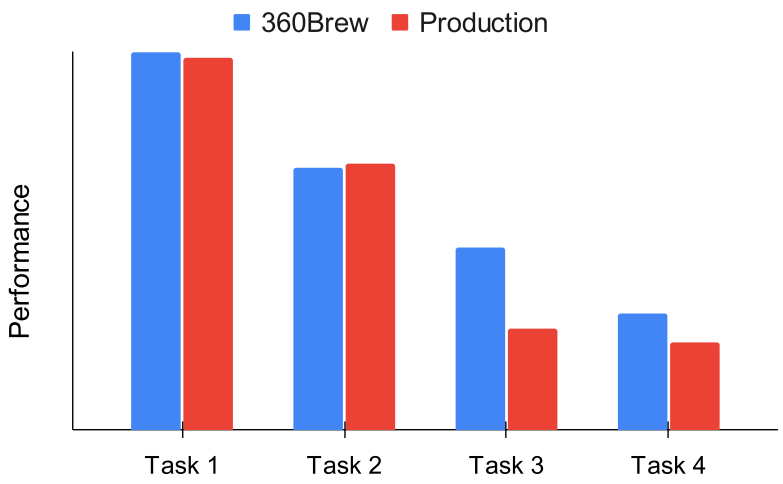

Figure 5: Performance on 4 T2 tasks across 4 surfaces which were not part of *360Brew* training.

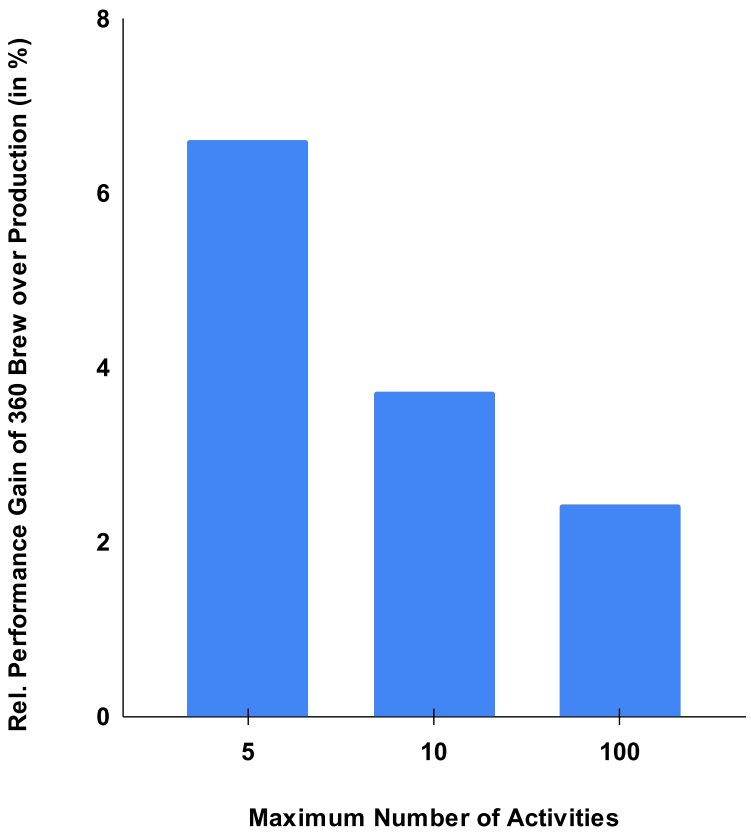

Figure 6: Relative performance gap between *360Brew* and production model as a function of max number of member interactions. For the members with a lower interaction count, *360Brew* performs better and has a larger gap with the baseline.

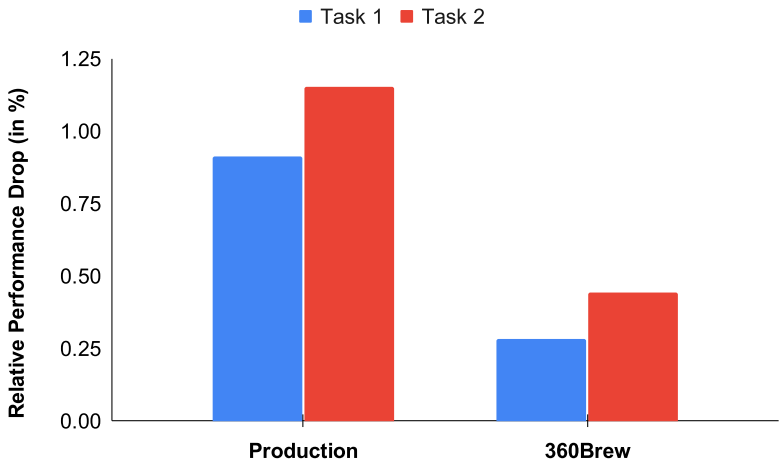

Figure 7: Performance of *360Brew* model compared to baselines as the test data gets temporally farther from the training data.

approaches like DeepSpeed ZeRO-2 and ZeRO-3 [33], which use data parallelism and ephemeral model parameters to keep memory utilization low on each node.[5]

While 3D parallelism showed a slight advantage in some experiments, **operability was the deciding factor**. Megatron-LM has a large dependency footprint, requiring C++ dependencies compiled with the C++11 ABI and changes to model code for row- and column-parallel layers. NVIDIA-supplied Ubuntu-based images are incompatible with our internal OS and kernel versions, complicating dependency management. Moreover, 3D parallelism requires different checkpoints for each tensor-, pipeline-, and data-parallelism configuration, making it brittle in environments with variable resources. These factors, along with the required code changes, render it incompatible with popular open-source libraries like Hugging Face Transformers [34] and vLLM [35], increasing operational burden.

Therefore, we opted for DeepSpeed-family techniques that require configuration only, adjusting parameters as infrastructure and model architecture changed. Ultimately, we chose PyTorch-native FSDP over the DeepSpeed library to minimize our dependency footprint and leverage future integrations like DTensors and `torch.compile` from the PyTorch community.

3.1.5 Training optimizer and learning rate scheduler

We use the schedule-free [36] version of the Adam optimizer [37] since it resulted in slightly better performance compared to alternatives and did not require fixing the number of steps (training horizon) beforehand. In addition to schedule-free, we experimented with two alternatives: (1) cosine learning rate scheduler [38] (2) almost constant learning rate with warmup/annealing at the beginning/end of training [39]. The cosine learning rate is the most popular approach, but requires fixing/knowing the training steps/horizon beforehand and makes it very difficult to change the training horizon on the fly. This would put a strain on restarting the experiments or running them longer than expected. Compared to the almost constant learning rate, schedule-free performed slightly better and was more robust to hyperparameter choices.

It is worth mentioning that with schedule-free we had to change our check-pointing strategy to make sure everything is saved in full precision. Also, in the schedule-free optimizer the computed train loss does not reflect the exact loss for the model [36]. As a result, **we augmented the training loop with callbacks to periodically evaluate the actual averaged model's loss to better measure the quality of the model for debugging purposes**. This was crucial for making decisions relying on the training loss curve.

[5]Note that ZeRO-1 [33] is entirely compatible with 3D parallelism, as discussed in related works combining DeepSpeed with Megatron-LM.

3.2 Inference

3.2.1 Inference Engine

We experimented with two different setups for our inference engine.

1. A standard forward pass using PyTorch Lightning, which was extremely simple and relatively easy to maintain.
2. vLLM [35].

We found that naive autoregressive generation is prohibitively slow without KV-caching. Ultimately, we chose vLLM for various reasons, though we continue to evaluate open-source inference engines regularly. While vLLM poses challenges due to its heavy dependency footprint and rapid development pace, these are offset by strong community and industry support. Moreover, vLLM offers practical advantages: consistent inference logic across both offline and online/nearline use cases, efficient autoregressive generation with the same codebase, and flexibility in tuning interdependent parameters like batch size and context length. For example, since our models are compute-bound rather than I/O-bound, we can use larger dataloader batch sizes to process input data concurrently while specifying a smaller token-level batch size for the inference engine to avoid exceeding GPU memory limits.

Challenges with vLLM. In our initial benchmarking with vLLM, we observed slowdowns of up to 30–50% for ranking probability estimation (single-token generation) and unexpected performance drops compared to PyTorch Lightning. After profiling and debugging, we identified the following issues:

1. **Prefix caching:** The default prefix caching in vLLM (v0.4.3) blocked computation to swap out the KV cache, without benefiting our latency requirements. We disabled prefix caching to get around this issue.
2. **NCCL bottlenecks:** Switching from FSDP in Lightning to vLLM's tensor parallelism introduced inter-layer NCCL bottlenecks, which DeepSpeed and FSDP avoid by overlapping communication with computation. This issue persists and we hope the improvements in newer vLLM versions will mitigate the remaining $\approx$ 30ms per decoder layer overhead.
3. **`LogitsProcessor` and chunked prefill.** We use the `LogitsProcessor` interface to obtain the log probabilities of target labels (e.g., "apply" or "click"). However, the `LogitsProcessor` logic failed when a vLLM scheduler change caused sequences to exceed the maximum token count—an assumption not guaranteed by the API. Upgrading from v0.5.2 to v0.5.3 led to severe performance drops due to vLLM enabling unsupported chunked prefill at larger sequence lengths. Additionally, we encountered Python runtime issues with multiprocessing workers not shutting down gracefully in our offline batch-processing scripts. Nonetheless, the guided decoding support in vLLM may eliminate such challenges and provide a relatively stable approach for our use cases.

Takeaways. These challenges are inevitable in actively developed libraries, but the strong vLLM community has been a significant asset to our progress. Finally, Figure 8 shows inference throughput (tokens/s/node on A100 nodes) versus the model input context length. Using vLLM and our optimizations, the throughput reduction is not linear in $\log_2$(context length), as quadratic compute scaling would predict. This suggests the throughput performance is likely memory-bound rather than compute-bound.

3.3 GPU Utilization

One of the main observable metrics for ensuring that training and batch inference are efficient is GPU utilization. In training, since we use FSDP, we can adjust the data parallelism (DP) factor depending on the number of GPUs. We used 256–512 NVIDIA H100 GPUs [40] for most of our training needs, with a batch size of 2–4 per GPU. In both training and inference, to achieve high GPU utilization, we created custom offline data preprocessors and on-the-fly iterable dataloaders.

To maximize GPU utilization during training, we leveraged FlashAttention v2 [41] and disabled FSDP CPU offloading. Additionally, we employed fused AdamW implementations [42], which combine

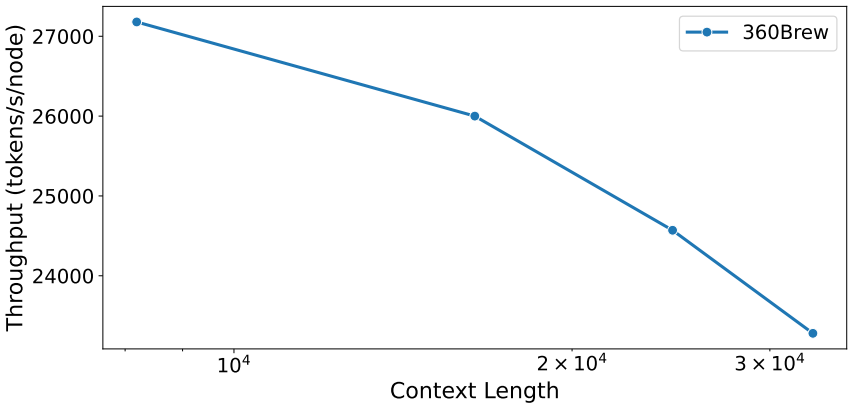

Figure 8: Effect of context length on batch inference throughput on Nvidia A100. Throughput decreases with increasing context length but not linearly with $\log_2$ (context length), indicating a memory-bound rather than compute-bound performance.

separate operators for optimizer states and model weight updates, enhancing computational efficiency. We also integrated Liger kernels [43] to reduce memory usage and accelerate training. The primary improvement from Liger is a fused kernel for cross-entropy loss calculation, which significantly reduces memory consumption—achieving a 30% reduction—and allows us to further increase the training batch size per GPU.

As mentioned before, for batch inference we use vLLM and find that it is able to maximize both High Bandwidth Memory (HBM) and compute utilization, reserving 85% of memory while still leaving enough for CUDA graph capture. Since vLLM on its own can effectively saturate a GPU, the single most important factor in utilization for inference has been avoiding "dead time" during which the GPU sits idle. While some of this is unavoidable—such as loading the initial model checkpoint into memory—we have sought to minimize other delays by mounting storage directly to the job container and caching data on the hosts to avoid lengthy downloads. As noted above, we apply our prompting logic on-the-fly as records are read. Although this significantly increases the data volume of those records, it allows us to free up memory and prevents GPUs from sitting idle during cumbersome preprocessing. We have empirically found that a dataloader batch size of 64 records with a maximum of $4 \times$ maximum sequence length of the batched tokens is sufficient.

Training follows a similar pattern: we minimize data loading and preprocessing time by employing iterable dataloaders and mountable storage with host-level caching, which eliminates severe bottlenecks. However, during training, our average compute utilization is inevitably lower as we scale horizontally and our training jobs become increasingly network-bound due to the need for all processes in distributed training to stay synchronized. In these instances, the availability of InfiniBand interconnects, such as those available in Azure, becomes critically important, as does the use of DeepSpeed ZeRO-family model/data parallelism to overlap compute with collective communications.

4 Data and Modeling Overview

4.1 Model Training Stages

We chose open-source pre-trained LLMs as our base models to leverage their general-purpose Internet knowledge and side-step the costly pre-training stage [7]. In particular, we use the Mixtral family of models [26]. Throughout the paper, if not specifically mentioned, the results correspond to the *360Brew* model built on top of Mixtral 8x22B, which is the largest model in the family. To ensure that the model adapts to information about the LinkedIn Economic Graph and can answer questions related to LinkedIn tasks, we use multiple stages of training, depicted below in Figure 9. We generally divide these stages into Continuous Pre-Training and Post-Training and discuss their details, challenges and benefits in the following sections.

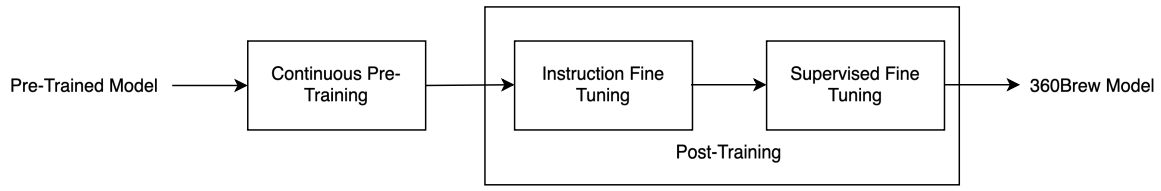

Figure 9: Different stages of training the *360Brew* model.

4.2 Data

360Brew 's input is standardized around natural language to provide exceptional flexibility in the usage of the model and easy adaptation to new recommendation tasks and product domains. Similar to previous work in the literature [7, 44], we leverage verbalization, similar to example in Table 1, to convert structured data, such as social graphs and interaction data, into natural language, allowing seamless integration into the *360Brew* model. Moreover, the data undergoes diversification, with verbalization being heavily varied through synthetic templates, structural ordering, and phrasing patterns to improve model memorization and generalization, as demonstrated in previous literature [45]. We cover the changes needed for the data in each training and evaluation stage in the following sections. Note that for *360Brew* , we only use data from the United States and primarily in English.

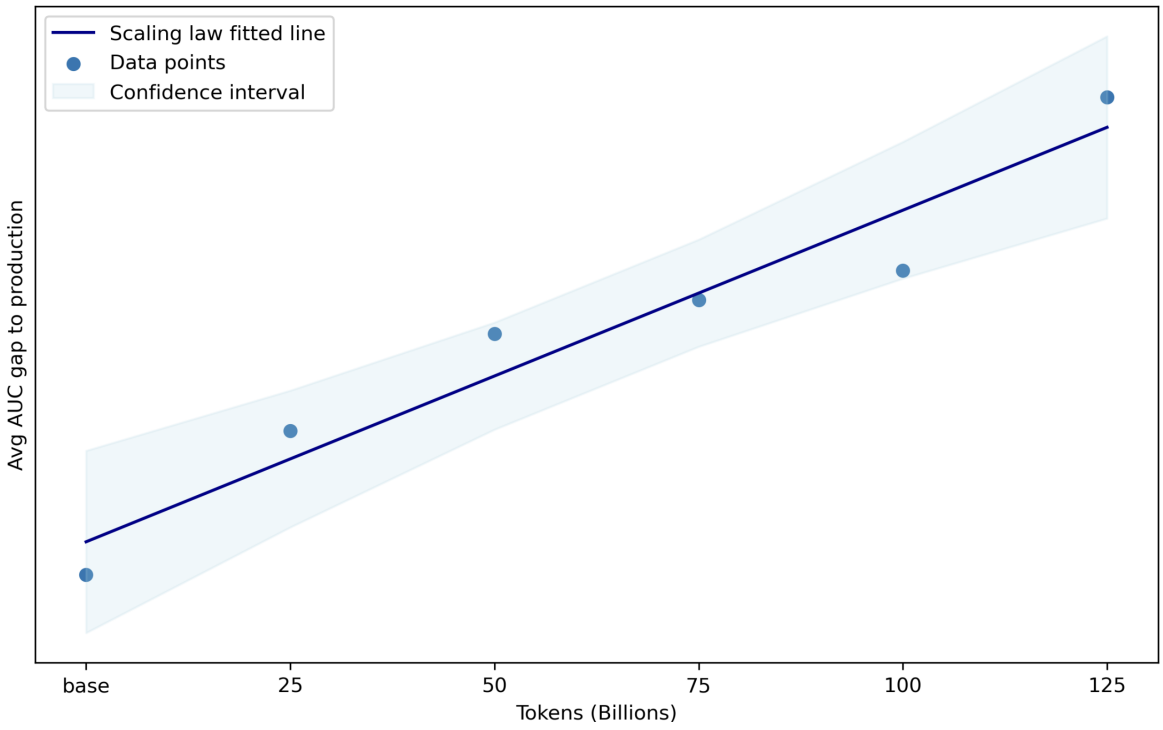

Figure 10: Closing the gap to production models on T1 tasks by scaling the token count during continuous pre-training.

4.3 Model Architecture and Size

As mentioned above, we built the *360Brew* model based on the open-source Mixtral family of models [26]. A major benefit of this family is that the larger models (8x7B and 8x22B) are both Mixture-of-Experts (MoE) models [46, 26], which facilitate efficient training and inference, while improving performance (see Figure 3).

4.4 Benchmarking & In-Context Learning

After the training phases, the *360Brew* model was benchmarked on T1 (in-domain) and T2 (out-of-domain) tasks. As shown in Figure 2, our model outperforms the baseline production models in their offline metrics on T1 tasks, whose historical distribution was part of its training. Moreover, Figure 5 shows that on T2 tasks, which are out-of-domain, we are competitive with the production models in their offline metrics without any additional training. This demonstrates that the *360Brew* model is capable of performing new tasks without further training, relying on in-context learning.

Note that while we trained the *360Brew* model on data from the United States and primarily in English, the evaluation data for each task is general and includes data from all locales. This means our model generalizes well to other languages and localities and can outperform production models that are trained on all localities. Thus highlighting its versatility.

During training we add augmentations to ensure the model is robust to different processing of the history, while during evaluation we use such processing as a lever to obtain more optimized results for the problem. For example, we use similarity-based, priority-based, and chronological history selection and ordering depending on the task. Note that these modifications do not change the model itself but only alter the prompting strategy for the model. For more details, see subsection 7.1.

5 Continuous Pre-training (CPT)

5.1 Pre-training Data

Our data comes from diverse sources and this section provides a general overview of the data we use from the LinkedIn Economic Graph[6].

- Member interactions with different LinkedIn products and surfaces, such as job applications, articles on the feed, notifications, search clicks, etc.
- Raw entity data such as member profiles, job descriptions, feed posts, articles, etc.
- Network data such as member-to-member connections and members following influencers.

Our training data covers 3–6 months of interactions from around 45 million sampled monthly active members. Since our proprietary data is heavily structured, following the lead from previous works [45], we added many augmentations to ensure that the information absorbed by the model about the members and items is extractable.

We used a fixed maximum context length of 16K for our CPT. To improve the training efficiency on our data, we used packing [47] in our training by combining multiple shorter examples to fill the context.

5.1.1 Interaction Encoding

One unique aspect of our data, often not encountered in previous literature on LLMs, is the need to encode historical interaction and social network data into text. We experimented with different ways of encoding such data, and the setup that worked best was to use member profiles, historical entities, and actions on those entities written out as text (similar to Table 1 but without the Instruction and Question). In our CPT data, we use chronological order to fill the history. Note that we also downsample (random sampling) some of the most prevalent interactions, such as 'impressions' in the history, since they do not carry too much information.

5.1.2 Data Recipe

We have around 15 data sources that are combined to build the training data for the *360Brew* model. Each of these data sources has its own characteristics in terms of token distribution, data quality, and information about the LinkedIn Economic Graph. For example, datasets about member activity on the feed are very noisy, as members' short-term interests change quite rapidly. Additionally, feed clicks and likes are particularly noisy signals. Therefore, how we mix these data sources has a large impact

[6]Note that our training data is limited to the United States and is mostly in English.

on the model's knowledge and ranking capability for LinkedIn ranking tasks. Furthermore, we follow data minimization principles, to identify data sources that are necessary for model training.

We leverage stratified sampling and importance sampling techniques to identify the optimal mix of these datasets that provide the highest value for downstream T1 and T2 tasks. To scale up experimentation and be mindful of computational budgets, the hyperparameters in stratified sampling and importance sampling are tuned on smaller models (Mistral-7B or Mixtral-8×7B) and extrapolated for larger models.

We have implemented numerous privacy-enhancing processes and techniques, such as minimizing personalized data (e.g., excluding email addresses and telephone numbers from the data) to help ensure that individual members' confidential information cannot be identified from the data.

5.2 Benefits and Downsides of CPT

5.2.1 CPT Data Scaling Law

CPT is the most data-intensive part of our experiments and is justified by the downstream improvements in model performance by scaling the number of training tokens (see Figure 10).

5.2.2 Better Understanding of the LinkedIn Economic Graph

To demonstrate that the CPT phase leads to a better understanding of the LinkedIn economic graph, we use a proxy metric and measure the ability of the model to complete the information about members and items it has seen during CPT. To this end, we built an evaluation flow to measure the rouge score (precision and recall) on completing the member profile, interactions and items it has seen as part of CPT. In particular, we give the model the first 50 tokens of some random information in CPT and prompt it to fill in the rest. We periodically measure such capability as we continue to train the model on more CPT tokens. Figure 11 shows the improvements in our evals as training progresses.

5.2.3 Context Length Sensitivity

One downside of using 16K maximum context length throughout our CPT was that our model became sensitive to context length and would not perform as well for very long contexts. To figure out at what context length the model would break, we used a "needle in a haystack" evaluation and ran it on our base model and *360Brew* model. While it is possible to fix this issue by gradually increasing context length during training [48], such an approach is resource-intensive. In subsection 7.2 we discuss solutions to this issue during inference, i.e. during the evaluation/benchmarking, which is much cheaper (but possibly not optimal). We also show that the needle in the haystack failure point gives us a good estimate of where the model performance would degrade in terms of task performance.

6 Post-training

While CPT allows the model to obtain fine-grained knowledge about the LinkedIn Economic Graph, i.e., our members and items as well as their interactions, the CPT-only model has the following limitations to solve the diverse set of tasks for our use cases:

- Lack of instruction following capability limits us from effectively prompting the model for out-of-domain tasks.
- The model is not robust to different historical interaction distributions (limited to chronological ordering), and variations in task labels. As a result, its performance would highly depend on pre-determined labels and history distributions.

The post-training phase addresses these shortcomings and comprises two stages: (1) An instruction-following stage to improve the instruction-following capabilities of the model, (2) Supervised Fine-Tuning (SFT) to make the model more robust to history distributions and task label changes.

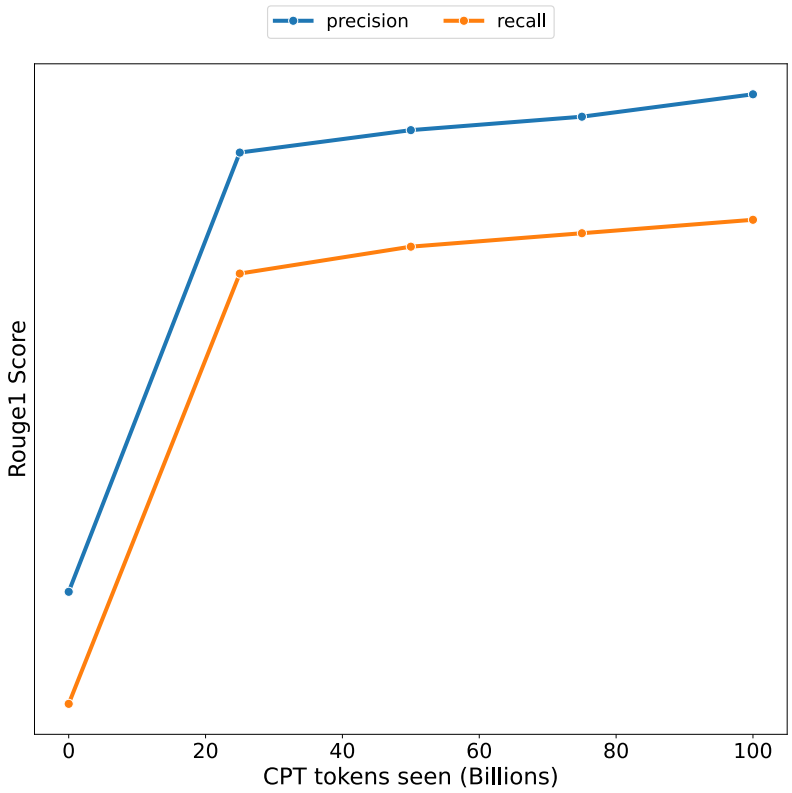

Figure 11: *360Brew* model's ability to recall the information in CPT. Each line shows different rouge measure metrics (precision, and recall) on different model checkpoints.

6.1 Instruction Fine-Tuning (IFT)

To enhance the model's ability to adapt to tasks not seen during training (both pre-training and supervised fine-tuning), we leverage instruction fine-tuning [49] to enhance zero-shot and few-shot generalization on unseen tasks. We leverage a blend of open-source and proprietary data to train for instruction-following. To this end, we leverage preference alignment algorithms like DPO [50] and ORPO [51]. Our training data for this is a blend of 1) Open-source data, i.e. a subset of UltraChat [52], 2) Internally generated instruction following data: we created this data by using LLMs to come up with questions and answers relevant to LinkedIn tasks and then judging the answers to find high quality and low-quality ones.

We measure the performance on open source benchmarks like IFEval [53] to quantify the effectiveness of our instruction tuning phase. We saw significant improvement (5% and more) in IFEval scores (across both prompt-level and instruction-level metrics) after training the model to follow instructions.

6.2 Supervised Fine-Tuning (SFT)

In the SFT phase, we train the model to learn member-entity interactions and improve its ability to predict member behavior when conditioned on specific surfaces. More importantly, this phase allows the model to leverage the knowledge gained during pre-training and predict member behavior based on the historical interactions available in the context.

6.2.1 Multi-Turn Chat (MTC) Template

We design our SFT prompts to employ a *Multi-Turn Chat (MTC)* format (illustrated in Table 2), and simulate a natural language conversation via alternating 'member'/'assistant' roles in the prompt.

While the instruction differs according to the task at hand, we ensure that the member/entity information aligns with the format used during pre-training. Furthermore, the format allows us to mask assistant responses and facilitate the model to capture the action patterns via supervision.

Instruction:
You are provided a member's profile and a set of [entities], their description, and interactions that the member had with them. Your task is to analyze the member-[entity] interactions so far and predict what action from the list of possible actions: $[Action_1, \cdots, Action_T]$ the member will take on the last given entity.
Member Profile: [member information]

[entity type] Interactions:
<member> What action will the member take on the following [entity]: [entity information]?
<assistant> $Action_1$
...
<member> What action will the member take on the following [entity]: [entity information]?
<assistant> $Action_K$

Table 2: A template for the MTC prompt during supervised fine-tuning.

6.2.2 Training Objective and MTC Loss

Given an MTC prompt, the Prompt loss ($\mathcal{L}_p$) is the auto-regressive next token prediction loss over the entire prompt computed using cross-entropy. The MTC loss ($\mathcal{L}_{mtc}$) is the masked variant of $\mathcal{L}_p$ over the assistant responses. The SFT loss ($\mathcal{L}_{SFT}$) is a weighted combination of ($\mathcal{L}_p$, $\mathcal{L}_{mtc}$) as follows:

$$\mathcal{L}_{SFT} = w_p * \mathcal{L}_p + (1 - w_p) * \mathcal{L}_{mtc}, \tag{3}$$

where $w_p \in [0, 1]$ denotes the prompt loss weight. Intuitively, $\mathcal{L}_p$ allows the model to continue its pre-training style learning over the new entities presented in the prompt, whereas $\mathcal{L}_{mtc}$ nudges it to focus only on the desired assistant response tokens. In the extreme case of $w_p = 0$, the model is being trained to purely predict the assistant tokens, resulting in a pure classifier model. However, a small $w_p > 0$ value allows the model to retain its broader text generation capabilities while learning to classify over complex historical action distributions. In our experiments, we found $w_p = 10^{-2}$ to work relatively well for our use case.

6.2.3 Data and Recipe for Member-Entity Interactions

We use different approaches to format and order the historical interactions in the MTC prompts. We found that these augmentations helped the model generalize better at the test time.

- **Chronological history:** Given a target entity and its timestamp, the historical interactions are selected from all the member activities before this timestamp and ordered chronologically.
- **Round-Robin history:** Given a pre-defined priority dictionary over all possible member actions, the historical interactions are first grouped by their corresponding actions and then selected in a round-robin fashion based on the priority values. This approach ensures a more class-balanced historical example for the model.
- **Similarity-based history:** Given a target entity, the historical interactions are ordered based on a text similarity metric in ascending order and then fit into the context. We use a combination of similarity scores and time stamps to order the examples.
- **Entity truncation:** For a fixed context length of the entire prompt, we truncate the structured entity information based on a fixed token budget. Thus balancing sufficient information per interaction and the total number of interactions we can fit into the prompt.
- **Action/label diversity:** A subset of our prompts are formatted to use augmented versions of the action labels to introduce task label diversity. This prevents the model from over-fitting to

a subset of action tokens and facilitates better ICL performance. This is important since the actions that our model needs to predict might be derived actions based on underlying member's behavior. For example, actions called click might correspond to different interactions in the data.

In our training recipe, we used a max context length of 16K. We believe there is more room for improvement here if we go to longer contexts. During our SFT training we used 10K-25K examples per surface and trained the model for 1 or 2 epochs. We found that scaling up the examples per surface beyond this or training more epochs does not bring any additional gains.

7 In-Context Learning

Previous works have shown that transformers can behave like few-shot learners that can model patterns given to them in the form of in-context examples [54, 22]. We believe this is a good framework to think about our *360Brew* model when applied to ranking problems. Intuitively, the model needs to personalize to the given member by analyzing their profile and behavior with different items and deciding how the member will behave next. In this framing, the only input to the model is the member's information and the member's behavior exemplar. But for most members, we often have many interactions while the context length of the model is limited (due to its training) and test time compute/memory/latency constraints. As a result, it is vital to utilize this resource (context length) efficiently for members with many historical interactions (most members). In order to best utilize the given context, we need to sample member's historical interactions when forming the ICL exemplar, depending on the problem. Note that, this could also be used as a flexible lever to better tune the model to a use case without fine-tuning it, e.g. forcing the model to look at the latest or most relevant examples.

7.1 ICL Recipe

Personalized model through ICL. For a given MTC prompt with K member-entity interactions, we can treat the prompt as a collection of K-tuples of $(m, e_i, a_i), i \in [K]$ as a dataset generated using inputs: member: m, entity: e_i and the outputs $a_i = f(m, e_i)$. Here $f \in \mathcal{F}$ is a function that models the behavior of a member, based on a complex non-linear function class $\mathcal{F}$. In this setup, we note that $\mathcal{L}_{mtc}$ resembles the masked regression loss [54, 22] that has been used to train transformers on synthetic regression tasks. This shows a key advantage of $\mathcal{L}_{mtc}$: it improves the robustness of the model to predict actions when presented with varying numbers of interactions (information about the member). In addition, this shows that the ICL is a way to create a personalized model for each member based on their profile and unique history.

How to enhance ICL capabilities of the model? Based on the philosophy mentioned above, we did extensive experimentation on the possible ways to perform ICL and get the best performance out of the model for different problem domains [7]. One of the approaches we took was to increase the number of entities while shortening the number of tokens used per entity. This was especially beneficial for tasks where having more examples with short textual information was sufficient.

We also experimented with different approaches to filling up the history. Specifically, we experimented with 3 different history constructions:

- **Chronological:** sorting the examples in ICL based on their timestamp from oldest to newest. This approach combined with the downsampling of some dominant labels is the one that gives the best results in most use cases.
- **Priority-based deduplication:** Since some of the examples can have multiple actions on them, we found out that in some cases it is worth doing deduplication on these actions based on some pre-defined priority. These priorities could come from the expert knowledge of these tasks. We found out that this approach helped some of the tasks where the target label conflicts with other labels frequently.

[7] The pre-cursor to all these experiments is to ensure that the model is robust to such changes in ICL history which we achieved by doing history augmentations and using MTC loss during SFT.

- **Similarity-based ICL:** We also experimented with similarity-based ICL that takes the target entity, finds similar items in history and uses them to fill in the ICL context. We found that this approach helps tasks that have diverse and nuanced interactions.

7.2 Long Context

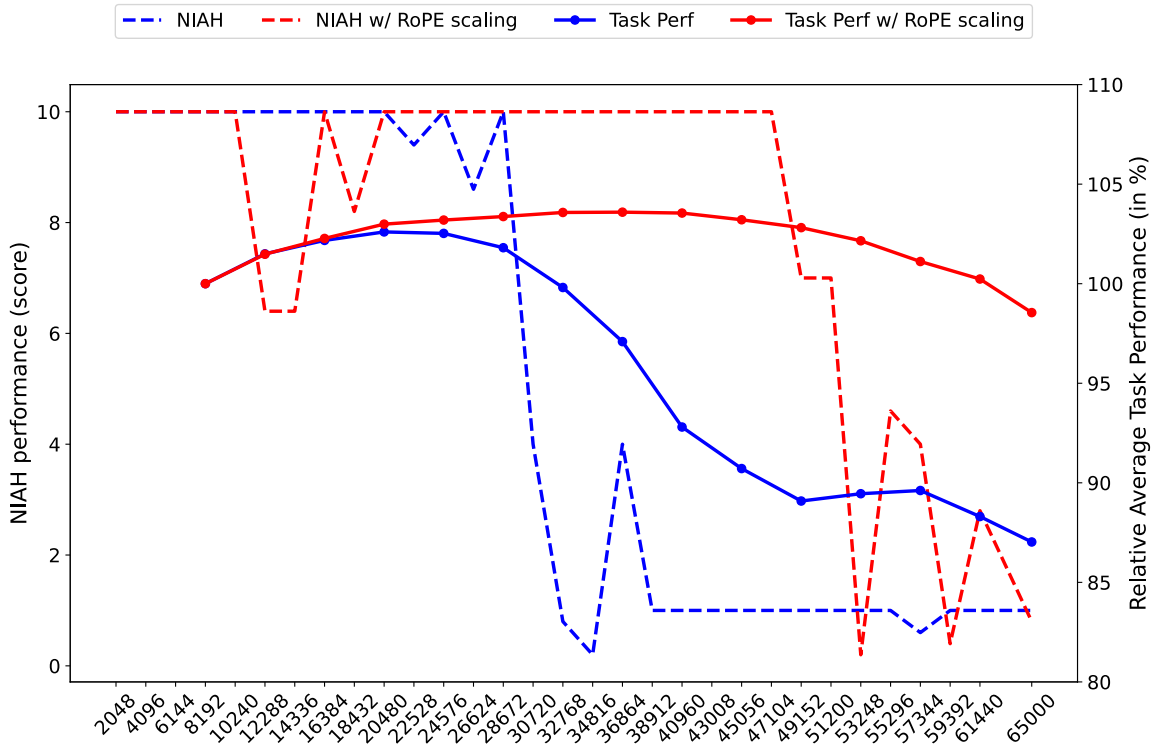

Figure 12: The dashed lines show the Needle-in-a-haystack (NIAH) score of the *360Brew* model (y-axis on the left) before/after (blue/red) inference time RoPE scaling. The solid lines depict the average task performance of the *360Brew* model as a percentage of its performance on 8K context length before/after (blue/red) inference time RoPE scaling. As depicted, the inference-time RoPE scaling (red) can improve the long context generalization of *360Brew* model after large-scale CPT. The NIAH score (easy to measure) is a good proxy to detect the length generalization of the model on task performances (hard to measure).

We implemented dynamic prompt filling where larger context lengths allow to pack in more information per entity and/or more historical actions. Higher context yields better results, since the model has more information available. Our base model, i.e. Mixtral-8x22B [26], was pre-trained up to maximum context length of 64K. But for efficiency, we only use a context length of 16K for CPT. In this case, we discovered that with increasing CPT data volume, our long context generalization deteriorates.

7.2.1 Measuring Context Length Generalization

To evaluate the context length generalization of the model, we used two evaluation tasks:

- **Needle-in-a-haystack (NIAH)**[8]: is based on retrieving target tokens (needles) presented to the model in long context prompts (haystacks) and asking the model to retrieve those needles. The model is then scored (0-10) on its ability to retrieve the needle from different parts of the haystack. This evaluation is fast and only requires the model to follow instructions.

[8] https://github.com/gkamradt/LLMTest_NeedleInAHaystack

- **Task-based performance on long context**: We evaluate the model on tasks with longer contexts by increasing the text per entity or the number of historical interactions to track its performance. Although this is closer to our final goal, it is computationally expensive.

We ran these evaluations on our model in Figure 12 (blue lines) and as one can see, the model performance drops significantly both on NIAH and internal evaluation beyond 30K context length even though the original base model supported 64K context length.

One major takeaway from these experiments is that we can utilize NIAH evaluation (which is fast) to approximate the context length generalization of the model. Moreover, the results from the test correlate well with the results from the model's performance on our tasks.

7.2.2 Inference-time RoPE scaling

To remedy the context length generalization issues, we considered several directions.

- Context length diversification during training.
- Additional post-training for context length extension, see for example YaRN [55] or LongROPE [56].
- Inference time context length extension (see [57] for an overview).

Increasing context length during any training stage is expensive since context length extension requires more GPUs to maintain the same training speed. Thus, from a pipeline perspective the later we can add the context length generalization the better. As a result, we opted for the last option which only changes the inference time mechanics of the model. One technique to extend context length generalization is to modify the positional embeddings (PE). Intuitively, we squeeze the PE into a subspace that the model has seen during training. There are different techniques on how to apply PE transformations [57] that can be applied during training and/or inference.

NTK-aware RoPE scaling. Experimentally, we have found that NTK-aware dynamic RoPE scaling where the rotary embeddings are dynamically scaled to maintain a stable Neural Tangent Kernel (NTK) alignment[9] works well if only applied at inference time, without any retraining. The parameters of interest are as follows:

1. RoPE base θ that controls the rotation frequency. Increasing θ decreases the frequency of the rotary period.
2. The original context length $T_{pretrain}$.
3. Scaling factor α which guides the maximum $T_{max} = T_{pretrain} * \alpha$.

We performed a grid search using the previously mentioned context length evaluations and experimentally found that setting $T_{pretrain} = 32,768$ and $\alpha = 2$ and the default θ value works well to improve the context length generalization of our *360Brew* model. Compared to the original case, the results of applying RoPE scaling are visualized in Figure 12. The NIAH and internal task performance no longer drop drastically at 30K context length, and we achieve better results with a larger context. We still see performance degradation at a higher context length of 50K+. We believe that such a decline is due to a combination of the limited capabilities of the original base model (Mixtral-8x22B) and a possible increase of stale activity data in the prompt.

References

[1] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[2] D. Roy and M. Dutta, "A systematic review and research perspective on recommender systems," *Journal of Big Data*, vol. 9, no. 1, p. 59, 2022.

[9] Reddit post by member u/emozilla (accessed 2024/12/02)

[3] P. Li, S. A. M. Noah, and H. M. Sarim, "A survey on deep neural networks in collaborative filtering recommendation systems," *arXiv preprint arXiv:2412.01378*, 2024.

[4] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *Proceedings of the web conference 2021*, 2021, pp. 1785–1797.

[5] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain *et al.*, "Matryoshka representation learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 233–30 249, 2022.

[6] X. Ren, W. Wei, L. Xia, L. Su, S. Cheng, J. Wang, D. Yin, and C. Huang, "Representation learning with large language models for recommendation," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 3464–3475.

[7] J. Chen, L. Chi, B. Peng, and Z. Yuan, "Hllm: Enhancing sequential recommendations via hierarchical large language models for item and user modeling," *arXiv preprint arXiv:2409.12740*, 2024.

[8] J. Zhai, L. Liao, X. Liu, Y. Wang, R. Li, X. Cao, L. Gao, Z. Gong, F. Gu, M. He *et al.*, "Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations," *arXiv preprint arXiv:2402.17152*, 2024.

[9] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE international conference on data mining (ICDM)*. IEEE, 2018, pp. 197–206.

[10] J. Li, M. Wang, J. Li, J. Fu, X. Shen, J. Shang, and J. McAuley, "Text is all you need: Learning language representations for sequential recommendation," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 1258–1267.

[11] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[12] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[13] B. Perozzi, B. Fatemi, D. Zelle, A. Tsitsulin, M. Kazemi, R. Al-Rfou, and J. Halcrow, "Let your graph do the talking: Encoding structured data for llms," *arXiv preprint arXiv:2402.05862*, 2024.

[14] H. Firooz, M. Sanjabi, W. Jiang, and X. Zhai, "Lost-in-distance: Impact of contextual proximity on llm performance in graph tasks," *arXiv preprint arXiv:2410.01985*, 2024.

[15] B. Fatemi, J. Halcrow, and B. Perozzi, "Talk like a graph: Encoding graphs for large language models," in *The Twelfth International Conference on Learning Representations*, 2024.

[16] J. Andreas, "Language models as agent models," *arXiv preprint arXiv:2212.01681*, 2022.

[17] B. AlKhamissi, M. Li, A. Celikyilmaz, M. Diab, and M. Ghazvininejad, "A review on language models as knowledge bases," *arXiv preprint arXiv:2204.06031*, 2022.

[18] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[19] J. Zhang, R. Xie, Y. Hou, W. X. Zhao, L. Lin, and J.-R. Wen, "Recommendation as instruction following: A large language model empowered recommendation approach," *arXiv preprint arXiv:2305.07001*, 2023.

[20] S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang, "Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5)," in *Proceedings of the 16th ACM Conference on Recommender Systems*, 2022, pp. 299–315.

[21] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[22] Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei, "Transformers as statisticians: provable in-context learning with in-context algorithm selection," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023, pp. 57 125–57 211.

[23] R. Agarwal, A. Singh, L. M. Zhang, B. Bohnet, S. Chan, A. Anand, Z. Abbas, A. Nova, J. D. Co-Reyes, E. Chu *et al.*, "Many-shot in-context learning," *arXiv preprint arXiv:2404.11018*, 2024.

[24] R. Anil, S. Gadanho, D. Huang, N. Jacob, Z. Li, D. Lin, T. Phillips, C. Pop, K. Regan, G. I. Shamir *et al.*, "On the factory floor: Ml engineering for industrial-scale ads recommendation models," *arXiv preprint arXiv:2209.05310*, 2022.

[25] J. Zhai, L. Liao, X. Liu, Y. Wang, R. Li, X. Cao, L. Gao, Z. Gong, F. Gu, M. He *et al.*, "Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations," *arXiv preprint arXiv:2402.17152*, 2024.

[26] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[27] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. Casas, L. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models. arxiv 2022," *arXiv preprint arXiv:2203.15556*, vol. 10, 2022.

[28] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," Mar. 2019. [Online]. Available: https://github.com/Lightning-AI/lightning

[29] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023.

[30] Y. Zhang, C. Chen, Z. Li, T. Ding, C. Wu, Y. Ye, Z.-Q. Luo, and R. Sun, "Adam-mini: Use fewer learning rates to gain more, 2024a," *URL https://arxiv. org/abs/2406.16793*.

[31] FairScale authors, "Fairscale: A general purpose modular pytorch library for high performance and large scale training," https://github.com/facebookresearch/fairscale, 2021.

[32] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[33] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[34] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

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

[36] A. Defazio, H. Mehta, K. Mishchenko, A. Khaled, A. Cutkosky *et al.*, "The road less scheduled," *arXiv preprint arXiv:2405.15682*, 2024.

[37] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] I. Loshchilov, F. Hutter *et al.*, "Fixing weight decay regularization in adam," *arXiv preprint arXiv:1711.05101*, vol. 5, 2017.

[39] A. Kosson, B. Messmer, and M. Jaggi, "Analyzing & reducing the need for learning rate warmup in gpt training," *arXiv preprint arXiv:2410.23922*, 2024.

[40] NVIDIA Corporation, *NVIDIA H100 Tensor Core GPU Architecture*, 2022, white Paper. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet

[41] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[42] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[43] P.-L. Hsu, Y. Dai, V. Kothapalli, Q. Song, S. Tang, S. Zhu, S. Shimizu, S. Sahni, H. Ning, and Y. Chen, "Liger kernel: Efficient triton kernels for llm training," *arXiv preprint arXiv:2410.10989*, 2024. [Online]. Available: https://arxiv.org/abs/2410.10989

[44] B. Fatemi, J. Halcrow, and B. Perozzi, "Talk like a graph: Encoding graphs for large language models," *arXiv preprint arXiv:2310.04560*, 2023.

[45] Z. Allen-Zhu and Y. Li, "Physics of language models: Part 3.1, knowledge storage and extraction," *arXiv preprint arXiv:2309.14316*, 2023.

[46] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[47] H. Ding, Z. Wang, G. Paolini, V. Kumar, A. Deoras, D. Roth, and S. Soatto, "Fewer truncations improve language modeling," *arXiv preprint arXiv:2404.10830*, 2024.

[48] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[49] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2021.

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

[51] J. Hong, N. Lee, and J. Thorne, "Reference-free monolithic preference optimization with odds ratio," *arXiv preprint arXiv:2403.07691*, 2024.

[52] N. Ding, Y. Chen, B. Xu, Y. Qin, Z. Zheng, S. Hu, Z. Liu, M. Sun, and B. Zhou, "Enhancing chat language models by scaling high-quality instructional conversations," *arXiv preprint arXiv:2305.14233*, 2023.

[53] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou, "Instruction-following evaluation for large language models," *arXiv preprint arXiv:2311.07911*, 2023.

[54] S. Garg, D. Tsipras, P. S. Liang, and G. Valiant, "What can transformers learn in-context? a case study of simple function classes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30 583–30 598, 2022.

[55] B. Peng, J. Quesnelle, H. Fan, and E. Shippole, "Yarn: Efficient context window extension of large language models," *arXiv preprint arXiv:2309.00071*, 2023.

[56] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, J. Xu, F. Yang, and M. Yang, "Longrope: Extending llm context window beyond 2 million tokens," *arXiv preprint arXiv:2402.13753*, 2024.

[57] Y. Huang, J. Xu, J. Lai, Z. Jiang, T. Chen, Z. Li, Y. Yao, X. Ma, L. Yang, H. Chen *et al.*, "Advancing transformer architecture in long-context large language models: A comprehensive survey," *arXiv preprint arXiv:2311.12351*, 2023.

A *360Brew* ***team members (alphabetical)***

Adrian Englhardt, Aman Gupta, Ben Levine, Dre Olgiati, Gungor Polatkan, Hamed Firooz, Iuliia Melnychuk, Karthik Ramgopal, Kirill Talanine, Kutta Srinivasan, Luke Simon, Maziar Sanjabi, Natesh Sivasubramoniapillai, Necip Fazil Ayan, Qingquan Song, Samira Sriram, Souvik Ghosh, Tao Song, Vignesh Kothapalli, Xiaoling Zhai, Ya Xu[10] , Yu Wang, and Yun Dai

[10]Now at Google DeepMind