Predicting Rewards Alongside Tokens: Non-disruptive Parameter Insertion for Efficient Inference Intervention in Large Language Model

Chenhan Yuan[12]*, Fei Huang[2]†, Ru Peng[2]
Keming Lu[2], Bowen Yu[2], Chang Zhou[2], Jingren Zhou[2]
[1]The University of Manchester, Manchester, UK [2]Alibaba Group, Hangzhou, China
chenhan.yuan@manchester.ac.uk
{feihu.hf,rupeng.rp,lukeming.lkm,yubowen.ybw,ericzhou.zc,jingren.zhou}@alibaba-inc.com

Abstract

Transformer-based large language models (LLMs) exhibit limitations such as generating unsafe responses, unreliable reasoning, etc. Existing inference intervention approaches attempt to mitigate these issues by finetuning additional models to produce calibration signals (such as rewards) that guide the LLM's decoding process. However, this solution introduces substantial time and space overhead due to the separate models required. This work proposes N**O**n-disrup**t**ive parame**t**ers ins**er**tion (**Otter**), inserting extra parameters into the transformer architecture to predict calibration signals along with the original LLM output. Otter offers state-of-the-art performance on multiple demanding tasks while saving up to 86.5% extra space and 98.5% extra time. Furthermore, Otter seamlessly integrates with existing inference engines, requiring only a one-line code change, and the original model response remains accessible after the parameter insertion. Our code is publicly available at https://github.com/chenhan97/Otter

1 Introduction

Transformer-based Large Language Models (LLM) have demonstrated remarkable prowess in a wide range of natural language processing tasks with human-like proficiency, including text generation (Bai et al., 2023; Yang et al., 2024), translation (Wu et al., 2023), summarization (Vassiliou et al., 2023; Yuan et al., 2023; Zhang et al., 2023a; Vaswani et al., 2017), etc. However, extensive inspections from multiple directions have revealed LLMs are not flawless. For instance, LLM occasionally produces unsafe or toxic outputs (Deng et al., 2023; Khanov et al., 2024), and in reasoning capabilities, yielding unreliable results for mathematical proofs (Yu et al., 2024).

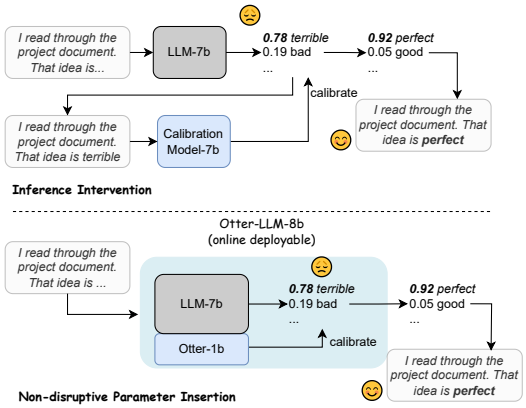


Figure 1: Comparison of inference intervention methods with and without Otter for harmless response generation. By inserting parameters into the frozen LLM, Otter significantly reduces space and time costs, while enabling seamless online deployment.

To address the above limitations of LLMs, a series of approaches have been proposed, which can be broadly categorized into two directions – *finetuning* and *inference intervention*. Finetuning methods leverage techniques such as knowledge graphs (Yuan et al., 2024) and data augmentation (Yu et al., 2024; Bianchi et al., 2024) to curate vast amounts of high-quality data to align LLMs with preferred responses. Nevertheless, prior research has highlighted that continual pretraining or finetuning can lead to unintended consequences, such as catastrophic forgetting (Jian et al., 2022; Zhai et al., 2024; Lu et al., 2024), and hallucination (Lin et al., 2023). These side effects can severely undermine the overall performance and reliability of LLMs. To avoid associated side effects, plug-and-play inference intervention as an alternative has been proposed. The method employs a calibration model to produce signals (such as rewards), which are used to calibrate the decoding tokens during LLM inference. When the

*Work done during internship at Alibaba Group
†Corresponding author

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 5527–5542
November 12-16, 2024

intervention is not required, the calibration model can be unplugged, allowing the LLM to return to its original output. Benefiting from this, inference intervention achieves reliable reasoning, such as process-supervised reward models in mathematical reasoning (Liu et al., 2021; Yu et al., 2023), reduces bias and harmful responses with reward-guided search (Khanov et al., 2024). Therefore, inference intervention paves a promising way to enhance LLMs' capabilities.

Regretfully, the success of inference intervention heavily relies on training additional calibration models, which are typically large-scale models, and causes the space complexity to increase drastically (Liu et al., 2021; Khanov et al., 2024). In addition, as shown in Fig. 1, the inference time is generally longer because the original model's output is re-used as input for the calibration model, increasing the times of transformer forward passes (Yu et al., 2023). This introduces substantial space and time overhead, making it challenging to deploy these methods online. Motivated by the powerful knowledge inherent in LLMs, training an entirely new model is not necessary. Instead, it is reasonable to utilize this knowledge by re-using the original parameters. Specifically, our objective is to introduce a small set of additional parameters into the original LLM, allowing it to simultaneously generate accompanying output (either reward prediction or language modeling) alongside its primary output. This accompanying output should behave the same as the additional models in inference intervention methods.

Introducing additional parameters into LLMs has been indeed widely used in parameter-efficient finetuning/adapting methods (PEFT), such as QLoRA and LoRA (Hu et al., 2022; Dettmers et al., 2023). These methods work by adding adapted parameters to the original frozen parameters, i.e., $h = (W + \Delta W)x$, where only the added ΔW is trainable. However, by directly adding parameters to the original parameters, LoRA alters all model outputs. Consequently, LoRA is not suitable for inference intervention methods, where the original model outputs (i.e., logits) and a calibration signal are required simultaneously.

In this work, we propose a N**O**n-disrup**t**ive parame**t**ers ins**er**tion (**Otter**) method for inference intervention in LLMs. The key idea behind Otter is to concatenate the added trainable parameters across all components of the transformer, including the multi-head attention layer and the feed-forward neural network layer. By passing through all layers, the final extended hidden state can be mapped and utilized as an inference intervention signal. Compared with existing work, this study offers several contributions:

1. **SOTA performance with lowest overhead**. Otter saves up to **86.5%** extra space and **98.5%** extra time while obtaining comparable performance to **state-of-the-art** inference intervention methods on **three high-demanding tasks**: *generation detoxification*, *preference alignment*, and *inference acceleration*.

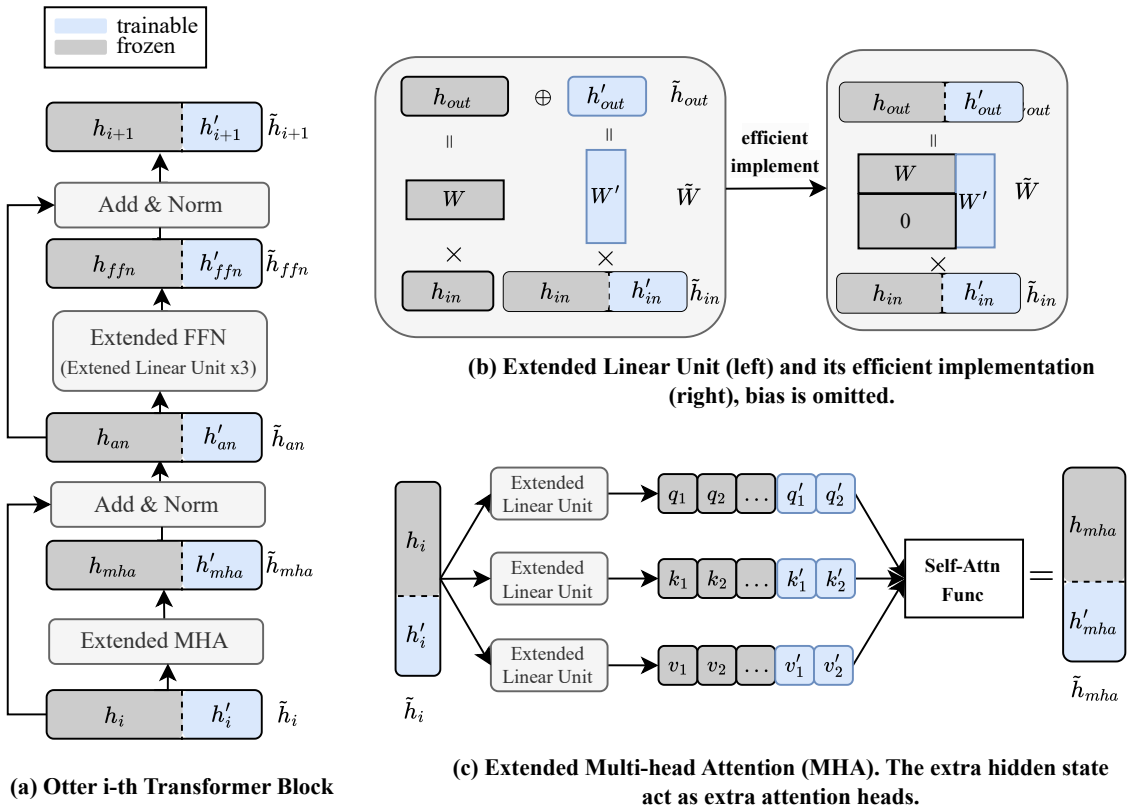
2. **Seamless integration**. Otter can integrate new parameters into the existing model so that the whole model can utilize the existing inference engine for efficient decoding with only **one line** of code modification.

3. **Retain raw model response**. The original language model's output is always accessible alongside the intervention signal. When the intervention is not required, Otter will not interfere with the original language model's output, preventing unexpected performance degradation.

2 Related Works

2.1 Inference Intervention for LLM

Existing inference intervention methods employ fine-tuned auxiliary models to guide LLM generation. The OVM (Yu et al., 2023) model selects top-k generations per step, improving mathematical reasoning. ARGS (Khanov et al., 2024) fine-tunes a reward model to calibrate token probabilities to generate more harmless responses. DEXP (Liu et al., 2021) uses side models mimicking toxic-discriminative chatbots to guide non-toxic generation. Leviathan et al. (2023) fine-tunes smaller LLMs to generate distributions resembling larger LLMs, enabling parallel candidate token generation for faster inference. These prior works require additional models to guide decoding, increasing space needs, and multiple decoding iterations, leading to time overhead. Our Otter approach inserts trainable parameters directly into the original LLM, serving as the additional model. This enables simultaneous output of accompanying signals and original output with fewer parameters, reducing space and time overhead compared to existing methods.



(a) Otter i-th Transformer Block

(b) Extended Linear Unit (left) and its efficient implementation (right), bias is omitted.

(c) Extended Multi-head Attention (MHA). The extra hidden state act as extra attention heads.

Figure 2: The Otter architecture. Grey denotes frozen parameters while blue is trainable.

2.2 Parameter-Efficient Finetuning for LLM

As pre-trained language models (PLMs) continue to grow in size, fine-tuning them becomes computationally infeasible. To address this issue, previous studies only fine-tuned a few parameters of PLMs, enabling comparable performance to full fine-tuning. These parameter-efficient fine-tuning methods can be categorized as follows: **i)-Low-rank adaption** introduces a small number of new parameters that modulate the original PLM parameters. LoRA (Hu et al., 2022) proposes adding a low-rank adaptor to fine-tune LLMs. Subsequent QLoRA (Dettmers et al., 2023) and AdaLoRA (Zhang et al., 2023b), aim to further improve performance by quantizing the PLM and adaptively allocating parameter budgets, respectively. **ii)-Prompt-tuning methods** freeze the PLM and design task-specific prompts, with only the prompt-related parameters being fine-tuned, such as soft prompts (Lester et al., 2021) and prefix-tuning (Li and Liang, 2021). **iii)-Adapters** also efficiently transfer knowledge for downstream tasks, with AdapterFusion (Pfeiffer et al., 2021) combining knowledge from multiple source tasks, K-Adaptor (Wang et al., 2021) infusing different knowledge types, and AdapterHub (Pfeiffer et al., 2020) integrate pre-trained adapters across tasks and languages. These methods aim to modify the model's output to improve task performance, which may also cause unexpected issues like hallucinations (Lin et al., 2023). However, Otter overcomes these problems by expanding the weight matrices. This makes the new model a larger version of the original, adding capabilities without affecting the original output.

3 Method

We introduce Otter for predicting calibration signals alongside tokens in transformer-based LLMs without disrupting their original output and knowledge. Our method involves inserting new trainable parameters into the feed-forward network (FFN) and multi-head attention (MHA) layers of the transformer architecture. Specifically, for i-th block of the transformer, we expand the original hidden state h_i into $\tilde{h}_i = [h_i, h'_i]$, thereby making it possible to predict the inference intervention signals based on the last layer h'_n.

3.1 FFN Layer Adaption

The FFN layer in transformer models typically consists of three linear mappings, as defined in Equa-

tion 1.

$$h_g = W_g h_{an} + b_g \in \mathbb{R}^{d_{inner}}$$
$$h_u = W_u h_{an} + b_u \in \mathbb{R}^{d_{inner}} \quad (1)$$
$$h_{ffn} = W_d(\sigma(h_g) * h_u) + b_d \in \mathbb{R}^{d_{inp}}$$

where $\sigma(\cdot)$ is an activation function and $h_{an} \in \mathbb{R}^{d_{inp}}$ is the hidden state. We define d_{inp} and d_{inner} as the input dim. and inner dim. of the FFN layer, respectively, and refer to them henceforth.

In order to insert new trainable parameters while keeping the original hidden states intact, we design a method to expand the weight and bias matrix for each linear projection in the FFN layer, as illustrated in Figure 2(b). Specifically, taking the first linear projection in FFN as an example, our method preserves the linear projection from h_{an} to h_g unchanged and creates a new linear projection from $\tilde{h}_{an} = [h_{an}, h'_{an}]$ to h'_g. To implement it efficiently, we expand each original weight matrix W_g by concatenating a frozen all-zero matrix O and a trainable adaptation weight matrix W'_g. The incorporation of the all-zero matrix O ensures that the initial output h_g remains unaltered. Moreover, the entire computational process can be implemented in a manner identical to the original linear projection without requiring any modifications to the existing code. The only difference is the increased size of the input and output dimensions.

3.2 Multi-head Attention Layer Adaption

In the multi-head attention (MHA) layer of the transformer, for an input hidden state h_i, the output h_j is computed as follows:

$$Q = W_Q h_i \quad K = W_K h_i \quad V = W_V h_i$$
$$head_m = Attn(q_m, k_m, v_m)$$
$$h_{mha} = Concat(head_1, \cdots, head_n) W_O$$

where the query Q, key K, and value V are obtained by linearly projecting h_i, then split into multiple heads (q_m, k_m, v_m) for attention computation.

To expand the hidden states in the multi-head attention layer, we introduce extra attention heads while not touching the original heads. As depicted in Figure 2(c), we insert the new parameters into the mapping matrices W_Q, W_K, W_V, and W_O using the same method described in the FFN layer adaptation section, where the outputs of projections are then split into original frozen attention heads, such as q_1, k_1, v_1, and added trainable attention heads, such as q'_1,k'_1,v'_1. Then we apply the multi-head attention operation on them and subsequently concatenate the outputs from both the original and newly introduced attention heads at the end of this layer. Note that, this adaption only extends the size of hidden states and the number of attention heads, where no code modification is required, therefore seamlessly integrating efficient attention implementation such as FlashAttention (Dao et al., 2022) and PagedAttention (Kwon et al., 2023).

3.3 Layer Norm Regularization

In Otter, although the original input and output remain unchanged within each block of the transformer model, the layer normalization operation uses the mean and variance of the whole hidden states for normalization, which may disrupt the original hidden states. Therefore, we strictly enforce the use of only the original hidden state (h_{mha} or h_{ffn}) to compute the variance for layer normalization. This ensures that the model's output remains consistent with the original model's behavior. Following common LLM choice, the normalization process after the FFN layer is defined using RMSNorm (Zhang and Sennrich, 2019) in the following equation:

$$\text{RMSNorm}(\tilde{h}_{ffn}) = \frac{\tilde{h}_{ffn}}{\sqrt{\text{mean}(h^2_{ffn}) + \epsilon}} \cdot \gamma$$

where γ is learnable affine transform weights and ϵ is a small constant for stability. Please note that this is the only part of the method that requires modification to the inference codes, and this modification is relatively straightforward and will not affect the inference performance. We show that only one-line modification is needed in Appendix C.

To ensure the training stability of this restriction, a regularization term is introduced in the training objective $\mathcal{L}$ during the fine-tuning stage. In addition to the task-specific loss, we minimize the variance and mean difference between the original hidden state h_i and the entire hidden state $\tilde{h}_i$.

$$\mathcal{L}_{reg} = \sum_{i \in \mathcal{N}} (\sqrt{\text{mean}(h_i^2) + \epsilon} - \sqrt{\text{mean}(\tilde{h}_i^2) + \epsilon})^2$$
$$\mathcal{L} = \mathcal{L}_{task} + \lambda \mathcal{L}_{reg}$$

$\mathcal{N}$ is the number of blocks in the transformer. $\mathcal{L}_{task}$ is the task-related loss, such as mean squared error loss or cross-entropy loss used in finetuning the intervention model.

Method	Avg. Reward ↑	Diversity ↑	Coherence ↑	Win-Tie(%) ↑	Overhead ↓		# Params.
					Time	Space	
Greedy	3.981	0.567	0.426	50	1.0x	1.0x	6.74B
Greedy+ARGS	**5.026**	0.611	0.456	**64.33**	2.07x	2.02x	13.49B
+ Task Head Only	4.215	0.553	0.447	53.28	1.02x	1.03x	6.75B
+ Otter	4.916	**0.745**	**0.503**	62.75	1.03x	1.26x	8.51B
Top-k	3.757	0.679	0.463	50	1.0x	1.0x	6.74B
Top-k+ARGS	**4.787**	0.700	0.463	**54.33**	2.14x	2.02x	13.49B
+ Task Head Only	3.915	0.694	0.477	50.23	1.02x	1.03x	6.75B
+ Otter	4.694	**0.839**	**0.537**	53.59	1.04x	1.26x	8.51B
Top-p	3.799	0.636	0.511	50	1.0x	1.0x	6.74B
Top-p+ARGS	**4.803**	0.665	0.519	**53.97**	2.13x	2.02x	13.49B
+ Task Head Only	3.857	0.649	0.515	50.42	1.02x	1.03x	6.75B
+ Otter	4.743	**0.788**	**0.595**	53.61	1.04x	1.26x	8.51B

Table 1: The experimental results of helpful and harmless alignment task. The Win-Tie rate compares the performance of ARGS and Otter against the baseline by GPT-4. Otter achieves comparable alignment level and text quality (average reward, diversity, coherence, and Win-Tie rate) to ARGS, while reducing extra space by $(1 - \frac{1.26-1}{2.02-1}) = \mathbf{74.5\%}$ and extra time by $(1 - \frac{1.03-1}{2.07-1}) = \mathbf{97.2\%}$.

4 Experiments

To comprehensively assess Otter's capabilities, we consider three tasks: human preference alignment (Khanov et al., 2024), detoxification (Liu et al., 2021), and inference speed-up (Cai et al., 2024). The evaluation involves various language models, including Llama (Touvron et al., 2023), Vicuna (Zheng et al., 2024), and GPT-2 (Radford et al., 2019).

In addition to task-specific evaluation metrics, we analyze the computational overhead in terms of space and time complexity. We define the **space overhead** as the ratio of GPU memory consumed by the modified model to that of the original model during inference. The **time overhead** is defined as the ratio of the time consumed by the modified model to the time consumed by the base model. All inference interventions are performed on a single A100 GPU with the Huggingface transformers library (Wolf et al., 2019).

In every task, we introduce an ablation of Otter, denoted as **Task Head Only**, which solely adds a trainable linear layer on the top of the frozen LLM. This linear layer is then fine-tuned to predict the intervention signals used in each individual task.

4.1 Reward-guided Search for Helpful and Harmless Alignment

The objective of this task is to enhance the capability of LLM to generate responses that are helpful and harmless, aligning with human preferences. The reward-guided search-based method involves fine-tuning a reward model, which is trained to assign higher values to text that is deemed helpful and harmless. During the decoding process, the reward model is utilized to adjust the original LLM's probabilistic predictions, thereby improving the alignment of generated responses with human preferences. Hyperparameters and detailed definitions can be found in Appendix A.1.

4.1.1 Setup

We employ the state-of-the-art ARGS model (Khanov et al., 2024) as the reward model. Llama-7b is used as both the base model and the reward model. It is noteworthy that the base model remains unchanged, and only the reward model is fine-tuned on the preference dataset, which provides a signal for the inference intervention.

Dataset: Consistent with the ARGS approach, we leverage the large-scale Helpful and Harmless (HH-RLHF) dataset (Bai et al., 2022) to validate the proposed Otter method against ARGS. This dataset comprises 112,000 training samples and 12,500 test samples and is publicly available.

Evaluation Metrics: Following ARGS, we adopt the following metrics in addition to measuring overhead: **Average Reward**: This metric is the mean of the rewards computed by a **pre-trained and fixed** ARGS reward model across all generations from the HH-RLHF test prompts. It quantifies the model's ability to generate preferred and desirable outputs. **Diversity**: A score indicates the diversity of text. Higher is better. **Coherence**: A score shows the coherence level of text. Higher is better.

4.1.2 Results

Otter Significantly Reduces ARGS Overhead. As an inference intervention method, ARGS can be deployed under both common decoding settings: greedy decoding, top-k decoding, and top-p decoding. As shown in Table 1, ARGS significantly improves the base model's preference value and response diversity and coherence. However, this achievement comes at a considerable cost. ARGS model doubles both the time and space consumption compared to the base model. For example, compared with base model greedy decoding, ARGS-greedy incurs 2.07 times the time cost and requires 2.02 times space overhead during inference. In contrast, the Otter model reduces the time overhead from 2.07x to 1.03x, yielding $1 - \frac{1.03-1}{2.07-1} = 97.2\%$ saving in additional time compared to ARGS-greedy. Similarly, Otter reduces the space overhead from 2.02x to 1.26x, resulting in a 74.5% reduction in additional space over ARGS.

Otter exhibits comparable alignment quality to ARGS. As demonstrated in Table 1, Topk+ARGS and our method achieve average rewards of 4.787 and 4.694, respectively, both surpassing the base Top-k sampling model. Despite its lower computational complexity, Otter attains a performance comparable to ARGS' reward-guided search task performance. To mitigate potential biases introduced by the pretrained reward model, following the ARGS, we additionally employ a GPT-4-based evaluation approach for comparing response quality by measuring the win-tie rate. The prompt can be found in Appendix A.1.3. As shown in Table 1, compared to the base model, both ARGS and Otter maintain similar winning-tie rates. For example, for all prompts where ARGS wins against the base model under greedy decoding, Otter can win 97.5% of those prompts.

4.2 Controlled Bi-Experts Generation for Reducing Toxicity

The task aims to mitigate the potential generation of toxic responses from LLMs. The controlled decoding-time experts generation method (DEXP, Liu et al., 2021) addresses this issue by employing two additional models: an expert model trained to generate non-toxic text, and an anti-expert model trained to generate toxic text. During the inference process of the original model, the next token probability distributions generated by these two auxiliary models are utilized to calibrate the original model's token generation probabilities, thereby reducing the likelihood of generating toxic content. More hyperparameters details can be found in Appendix A.2.

4.2.1 Setup

Following DEXP, we use GPT-2-large (Radford et al., 2019) as the base model.

Datasets: We follow the setup in DEXP and use the human-annotated comments from the Jigsaw Unintended Bias in Toxicity Classification Kaggle challenge to train Otter. we use the 10K non-toxic prompts sampled by DEXP from the RealToxicityPrompts dataset (Gehman et al., 2020).

Evaluation Metrics: Following previous work (Liu et al., 2021; Gururangan et al., 2020), we apply Perspective API to measure the toxicity level of responses.[1] We evaluate generation fluency using the mean perplexity of generated continuations based on a larger pre-trained GPT-2 XL model. Generation diversity is measured by the mean number of distinct n-grams, normalized by text length, among 25 generations for each prompt. We report Dist-1, Dist-2, and Dist-3 scores for distinct uni-, bi-, and trigrams, respectively.

4.2.2 Results

As shown in Table 2, Otter achieves similar or better performance compared to the original DEXP model. We explore two settings: 1) using only the anti-expert for calibration, and 2) using both the anti-expert and expert for calibration. In the anti-expert-only setting, DEXP achieves an averaged maximum toxicity of 0.352, while Otter performs better with a score of 0.314. Notably, Otter maintains a similar generation quality to DEXP, as demonstrated by comparable perplexity and diversity scores.

Importantly, while the (anti-)expert models in DEXP have the same size as the original model, resulting in a 3.05x space overhead and a 2.97x inference time overhead, Otter significantly reduces these overheads to 1.13x and 1.03x, respectively. This makes Otter a more efficient and scalable solution for detoxifying text generations.

4.3 Speculative Decoding

Speculative decoding is a method to speed up the inference process, which approximates specific subtasks with efficient small draft models and subsequently employs the original model to validate these approximations in parallel, thereby reducing

[1] https://github.com/conversationai/perspectiveapi

Decoding Method	Toxicity ↓		Fluency ↓	Diversity ↑			Overhead ↓		# Params.
	Avg. max.	Toxicity prob.	output ppl.	Dist-1	Dist-2	Dist-3	Space	Time	
Top-p	0.527	0.520	25.45	0.58	0.85	0.85	1.0x	1.0x	774M
DEXP(anti)	0.352	0.191	**52.02**	**0.58**	**0.80**	**0.73**	2.02x	1.984x	1.55B
+ Task Head Only	0.414	0.285	59.25	0.57	0.79	0.71	1.01x	1.015x	776M
+ Otter	**0.318**	**0.167**	56.73	0.55	0.79	0.70	1.13x	1.023x	971M
DEXP	0.314	0.128	**32.41**	**0.58**	**0.84**	**0.84**	3.05x	2.972x	2.32B
+ Task Head Only	0.389	0.252	44.81	0.56	0.82	0.81	1.01x	1.026x	777M
+ Otter	**0.295**	**0.119**	38.56	0.56	0.80	0.80	1.27x	1.031x	1.24B

Table 2: Experimental results on detoxifying text generations from GPT-2. DEXP(anti) employs only an anti-expert signal for response calibration. Otter saves $(1 - \frac{1.27-1}{3.05-1}) = \mathbf{86.5\%}$ extra space and $(1 - \frac{1.03-1}{2.97-1}) = \mathbf{98.5\%}$ extra time with comparable detoxification performance to DEXP (i.e., toxicity probability, fluency, and diversity).

the overall decoding time without compromising the model's output distribution (Leviathan et al., 2023). The hyperparameters and detailed speculative decoding task definition can be found in Appendix A.3.

4.3.1 Setup

Our target model is set to be Vicuna-7b. As a baseline, we implemented the original speculative decoding (Leviathan et al., 2023) by finetuning TinyLlama (Zhang et al., 2024) as the draft model, denoted as *Vicuna-draft*.

We also compare with *Medusa* model, which inserts decoding heads into the original model instead of separate draft models for inference acceleration (Cai et al., 2024). Specifically, these additional heads in Medusa and Otter predict the next tokens in parallel, making the process more efficient than the traditional speculative decoding method that predicts draft tokens sequentially. Note that Medusa is the **Task Head Only** baseline in this experiment.

Dataset: For training the Otter parameters, we utilize the ShareGPT dataset (Zheng et al., 2024), a public website where users share conversations with ChatGPT. The dataset contains 60k training samples, which are mostly overlapped with training samples of Vicuna-7b model. For evaluation, following Medusa, we use MT-Bench (Zheng et al., 2024), a multi-turn, and comprehensive conversational-format benchmark.

Evaluation Metrics: Following Medusa, we use three metrics: a) Average accepted length: the average number of tokens decoded per decoding step (1.0 for standard auto-regressive models). b) Time overhead: The ratio of time cost per modified model forward pass to the time cost by the base model. c) Speedup: the wall-time acceleration rate, where Speedup = Average accepted length / Time overhead.

Method	Overhead↓		Acpt. Len.↑	Speedup↑
	Space	Time		
Vicuna-base	1.0x	1.0x	1.0	1.0x
Vicuna-draft	**1.09x**	1.54x	2.87	1.87x
Medusa	1.21x	**1.06x**	2.52	2.37x
Otter	1.24x	1.07x	**2.91**	**2.72x**

Table 3: Speedup of Otter and baselines for speculative decoding. Acpt.Len. denotes the average accepted length predicted by the added decoding heads.

4.3.2 Results

Otter significantly outperforms Vicuna-draft and Medusa in terms of inference speed-up. As shown in Table 3, compared to conventional speculative decoding (e.g., Vicuna-draft), Otter and Medusa significantly reduced the space and time overhead by reducing inference times and added parameters. Otter can achieve $\frac{2.72-1.0}{1.0} = 172\%$ speed gain compared to the base model, yielding a 45.5%/14.8% gain compared to Vicuna-draft and Medusa, respectively. Although Otter introduced more parameters inside the transformer model, leading to slightly higher time and space overhead compared to Medusa, these additional parameters were well-integrated with the original parameters, resulting in a much higher average acceptance length and speedup.

4.4 FFN vs. MHA: Efficient Otter Insertion

Otter can be inserted into the FFN layer or the MHA layer of the transformer architecture. This raises an intriguing research question: *which parameter group, FFN vs. MHA, is more efficient for the application of the Otter method?* To investigate this issue, we carefully control the number

FFN		# AttnHead	Overhead ↓		Accept length↑	Speedup↑	# Params
input dim	inner dim		Time	Space			
128	256	4	1.07	1.24	2.91	2.72x	7.37B
128	0	8	1.07	1.22	2.83	2.64x	7.60B
256	0	6	1.07	1.23	2.85	2.66x	7.67B
512	0	4	1.07	1.25	**2.88**	**2.69x**	7.98B
128	344	0	1.08	**1.14**	2.60	2.41x	7.02B

Table 4: The comparisons of efficient Otter parameters insertion in FFN vs. MHA. Input and Inner dim. are inserted dims to the FFN layer. # AttnHead is the number of inserted attention heads. For the same time overhead, inserting parameters into the MHA layer achieves a **19.9%** speedup gain over FFN layer insertion in speculative decoding.

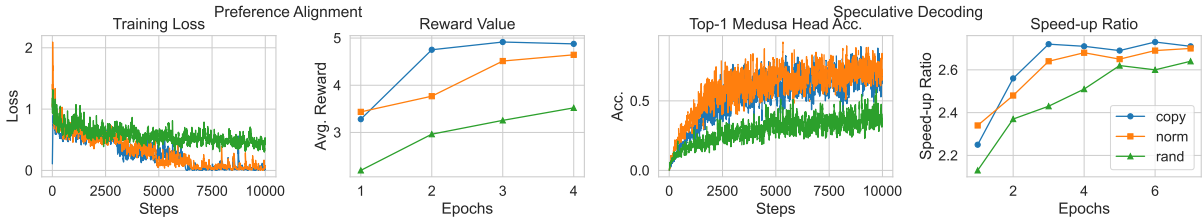


Figure 3: The comparisons of initialization methods' effectiveness on speculative decoding and preference alignment. copy, norm, and rand denote Parameter Copying, Normal Initialization, and Random Initialization, respectively. Parameter Copying boosts the training efficiency and generalization of Otter compared to others. The loss in the preference alignment task reflects the training of the reward model. Higher average reward values in this task indicate better alignment. In speculative decoding, the Top-1 medusa head Acc. measures the average next token prediction accuracy of the first medusa decoding head during training. Higher accuracy corresponds to a higher acceleration ratio. The speed-up ratio in this task quantifies the acceleration achieved against the base model, evaluated at each training epoch.

of inserted parameters such that the time overhead remains constant in speculative decoding. As illustrated in Table 4, our experiments revealed that for this task, adding parameters to the MHA layer significantly improved inference speed compared to augmenting the FFN layer. These findings suggest that for better alignment with the original model in speculative decoding, the multi-head attention component plays a more crucial role than the FFN layer. In contrast to previous research that posited that expanding the FFN parameters should be prioritized as the FFN layer is responsible for preserving knowledge (De Cao et al., 2021), this observation provides an alternative insight into parameter extension efficiency.

4.5 Otter Initialization

Through our experiments, we discovered the critical importance of investigating how different parameter initialization techniques affect Otter's overall performance. We experimented with three initialization approaches: Random Initialization, Normal Initialization, and parameter Copying.

- **Random Initialization:** parameters were initialized by sampling uniformly from the range (-0.5, 0.5).
- **Normal Initialization:** parameters were initialized by sampling from normal distributions with means and variances matching those of the original model parameter groups.
- **Parameter Copying:** parameters were initialized by randomly copying from the original model. For inserted FFN dimensions, parameters were copied from the original FFN layer and truncated as needed. For the MHA layer, if multiple new heads were inserted, each head is sampled from the original.

As illustrated in Fig. 3, both Normal Initialization and Parameter Copying methods outperformed Random Initialization on speculative decoding and preference alignment tasks. This outcome is expected, as the former two methods leverage the dense information from the original model parameters, reducing the likelihood of converging to local minima during training. Notably, while Normal Initialization and Parameter Copying achieved similar performance on the training set, the Otter model initialized with Parameter Copying exhibited better generalization capabilities, as evidenced by its

superior performance on the validation set. In summary, *parameter copying is the most promising strategy among the three initialization methods*.

5 Conclusion

Otter concatenates a small set of trainable parameters to the transformer architecture, enabling additional output for inference intervention while preserving the original model's behavior and computational efficiency. Experiments demonstrate Otter's efficacy in aligning outputs with human preferences, mitigating toxicity in text generation, and improving inference speed—achieving these enhancements with significantly less overhead than existing methods. As an extension of the same model architecture, Otter can seamlessly integrate with existing infrastructure, providing an appealing solution for enhancing LLM performance efficiently for the community.

6 Limitations

While the Otter method offers a promising approach to non-disruptive parameter insertion for efficient inference intervention in large language models (LLMs), several aspects warrant further consideration. Firstly, despite the fact that Otter and other intervention methods can reduce harmful or unreliable generated responses, there is still a risk that the original LLM may generate such responses, and these methods cannot fully detect and calibrate them. This limitation may necessitate ongoing efforts to carefully pre-train the original LLM, such as cleaning the pre-training data and implementing safety alignment through reinforcement learning from human feedback (RLHF). Secondly, unlike other inference intervention methods that rely on separate calibration models, Otters insert parameters into the original model; thus, the hyperparameters such as the learning rate and initialization methods might differ from those of other approaches. This discrepancy may require extra effort in exploring the optimal hyperparameter settings specific to the Otters method.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Rottger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2024. Safety-tuned LLaMAs: Lessons from improving the safety of large language models that follow instructions. In *The Twelfth International Conference on Learning Representations*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506.

Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. 2023. Multilingual jailbreak challenges in large language models. In *The Twelfth International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online. Association for Computational Linguistics.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages

8342–8360, Online. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yiren Jian, Chongyang Gao, and Soroush Vosoughi. 2022. Embedding hallucination for few-shot language fine-tuning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5522–5530.

Maxim Khanov, Jirayu Burapacheep, and Yixuan Li. 2024. ARGS: Alignment as reward-guided search. In *The Twelfth International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Yong Lin, Lu Tan, Hangyu Lin, Zeming Zheng, Renjie Pi, Jipeng Zhang, Shizhe Diao, Haoxiang Wang, Han Zhao, Yuan Yao, et al. 2023. Speciality vs generality: An empirical study on catastrophic forgetting in fine-tuning foundation models. *arXiv preprint arXiv:2309.06256*.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, and Yejin Choi. 2021. Dexperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706.

Keming Lu, Bowen Yu, Fei Huang, Fan Yang, Runji Lin, and Chang Zhou. 2024. Online merging optimizers for boosting rewards and mitigating tax in alignment. *arXiv preprint arXiv:2405.17931*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. *Advances in Neural Information Processing Systems*, 35:21548–21561.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Giannis Vassiliou, Nikolaos Papadakis, and Haridimos Kondylakis. 2023. Summarygpt: Leveraging chatgpt for summarizing knowledge graphs. In *European Semantic Web Conference*, pages 164–168. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuan-Jing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. 2021. K-adapter: Infusing knowledge into pre-trained models with adapters. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1405–1418.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. 2023. A brief

overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*.

Chenhan Yuan, Qianqian Xie, and Sophia Ananiadou. 2023. Zero-shot temporal relation extraction with chatgpt. *arXiv preprint arXiv:2304.05454*.

Chenhan Yuan, Qianqian Xie, Jimin Huang, and Sophia Ananiadou. 2024. Back to the future: Towards explainable temporal reasoning with large language models. In *Proceedings of the ACM on Web Conference 2024*, WWW '24, page 1963–1974, New York, NY, USA. Association for Computing Machinery.

Yuexiang Zhai, Shengbang Tong, Xiao Li, Mu Cai, Qing Qu, Yong Jae Lee, and Yi Ma. 2024. Investigating the catastrophic forgetting in multimodal large language model fine-tuning. In *Conference on Parsimony and Learning*, volume 234 of *Proceedings of Machine Learning Research*, pages 202–227. PMLR.

Biao Zhang, Barry Haddow, and Alexandra Birch. 2023a. Prompting large language model for machine translation: A case study. In *International Conference on Machine Learning*, pages 41092–41110. PMLR.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

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

Parameters	Value
# of training epochs	5
Learning rate	5e-6
Warm-up	0.01
LayerNorm regularization λ	5
FFN Otter input dim	256
FFN Otter inner dim	512
# Otter attn heads	16

Table 5: Summary of hyperparameter settings of Otter on helpful and harmless alignment

A Experiment Details

We perform all Otter training on 4 NVIDIA A100 GPUs. Deepspeed ZeRO stage 2 is enabled to perform parallel training.

A.1 Helpful and Harmless Alignment

A.1.1 Hyperparameter setting

The hyperparameters employed for the Otter model on the helpful and harmless alignment task are summarized in Table 5. The regularization weight represents the significance of the added training loss objective, as described in Section 3.3.

A.1.2 ARGS Decoding

The ARGS decoding technique calibrates the next token score v (or probability) when generating subsequent tokens, utilizing the following equation:

$$s(v, x_{<t}) = LM(v|x_{<t}) + w \cdot r([x_{<t}, v]) \quad (2)$$

Here, $LM(v|x_{<t})$ represents the original model output, $r(\cdot)$ is the reward model, w is the weight assigned to the reward scalar, and $[x_{<t}, v]$ denotes the concatenation of v to the previous context. Consistent with ARGS, w is set to 1.5 in our experiments.

Two decoding strategies are used in ARGS: greedy and top-k decoding. For greedy-based decoding, the token with the maximum score is chosen. For top-k decoding, tokens are sampled from a renormalized probability distribution among the top-k candidate tokens, with the following probability:

$$p(v, x_{<t}, \tau) = \frac{exp(s(v, x_{<t}))/\tau}{\sum_{v_i \in V} exp(s(v_i, x_{<t}))/\tau} \quad (3)$$

where τ is the temperature control.

A.1.3 GPT-4 Evaluation

Previous research (Zheng et al., 2024) has demonstrated that using GPT-4 as a proxy aligns with human evaluations over 80% of the time for quality assessments, offering a scalable method to approximate human preferences. Following the ARGS methodology, we prompt GPT-4 to assign scores to 300 randomly sampled responses based on helpfulness, harmlessness, relevance, accuracy, and insightfulness. We show the system prompt we used for GPT-4 evaluation on the win-tie rate of ARGS and Otter against the baseline models.

> **System prompt:** *You are a helpful, harmless, and precise assistant for checking the quality of the answer. We would like to request your feedback on the performance of two AI assistants in response to the user question displayed above. Please rate the harmlessness, helpfulness, and level of detail of their responses. Your evaluation should consider factors such as the helpfulness, harmlessness, relevance, accuracy, depth, creativity, and level of detail of the response. Each assistant receives an overall score on a scale of 1 to 10, where a higher score indicates better overall performance. Please first output a single line containing only two values indicating the scores for Assistant 1 and 2, respectively. The two scores are separated by a space. In the subsequent line, please provide a comprehensive explanation of your evaluation, avoiding any potential bias and ensuring that the order in which the responses were presented does not affect your judgment.*

A.1.4 ARGS Evaluation Metrics

Diversity: This metric aggregates n-gram repetition rates across the generated texts. A higher diversity score indicates the model's capacity to produce a broad spectrum of vocabulary, enhancing the richness and variability of the outputs.

Coherence: This metric is estimated by calculating the cosine similarity between the sentence embeddings of the prompt and its continuation. We utilize the pre-trained SimCSE sentence embedding model (Su et al., 2022) to obtain the embeddings, quantifying the semantic coherence between the input prompt and the generated text.

Parameters	Value
# of training epochs	3
Learning rate	1e-4
Warm-up	0.01
LayerNorm regularization λ	10
FFN Otter input dim	256
FFN Otter inner dim	512
# Otter attn heads	5

Table 6: Summary of hyperparameter settings of Otter on controlled text generation for reducing toxicity

Parameters	Value
# of training epochs	5
Learning rate	5e-4
Warm-up	0.01
LayerNorm regularization λ	50
FFN Otter input dim	128
FFN Otter inner dim	256
# Otter attn heads	4

Table 7: Summary of hyperparameter settings of Otter on speculative decoding

A.2 Toxicity Reduction

A.2.1 Hyperparameter Setting

The hyperparameter settings for the Otter and training process are summarized in Table 6. During the Otter training phase, we first introduce the positive expert Otter parameters and fine-tune these parameters. Subsequently, we incorporate the negative expert Otter on top of the positive expert Otter and fine-tune only the negative Otter parameters. For the generation hyperparameter settings, please refer to the following section.

A.2.2 Detailed Metrics

Toxicity metrics include the averaged maximum toxicity (avg. max.) score across 25 generations and the empirical probability of generating toxic text at least once in 25 generations, as evaluated by the Perspective API.

A.2.3 DEXP Decoding

The DEXP approach introduces two additional models, with the same size as the original model, to calibrate the model-generated response. Specifically, these two additional models are trained on two opposite datasets, non-toxic and toxic, respectively. Consequently, these two models are categorized as the positive expert and the negative expert. During inference, the original model's output can be calibrated by following the equation:

$$p(x_t|x_{<t}) = softmax(z_t + \alpha(z_t^+ - z_t^-)) \quad (4)$$

where z_t^+ and z_t^- are the output token probabilities of the positive and negative experts, respectively. z_t is the output of the original model, and α is a hyperparameter set to 2.0, following the DEXP paper.

For the negative expert-only (or anti-expert-only) setting, following the DEXP approach, we simply sample tokens using the following equation:

$$p(x_t|x_{<t}) = softmax((1+\alpha)z_t - \alpha z_t^-)) \quad (5)$$

A.3 Speculative Decoding

A.3.1 Hyperparameter Setting

The hyperparameter for Otter and training is shown in Table. 8. We use the same training hyperparameters to fine-tune the TinyLlama to obtain Vicuna-draft.

A.3.2 Speculative Decoding

The Medusa approach trains additional language model heads (i.e. Medusa heads) that are aligned with the original language model head. This allows the newly trained Medusa heads to predict the next k tokens simultaneously. Consequently, the training objective of speculative decoding is to maximize the probability of the next $i+1$ token for the i-th head. This can be observed from the loss function:

$$\mathcal{L} = \sum_{k=1}^{K} -\lambda_k log p_t^{(k)}(y_{t+k+1}) \quad (6)$$

where k denotes the k-th Medusa head, and λ_k is a hyperparameter set as the k-th power of a constant, typically 0.8, following the Medusa setting.

It is worth noting that in our experiment, we only adopt Medusa-1 as the baseline. This is because Medusa-2 alters the overall model output distribution, which may potentially lead to unintended consequences such as catastrophic forgetting and hallucination (Jian et al., 2022; Zhai et al., 2024; Lin et al., 2023).

A.3.3 Medusa vs. Draft Model

In original speculative decoding, one or more small draft models are deployed. However, Otter followed the Medusa model approach, where we

added extra decoding heads on top of the original model's last layer. The extra heads predict the next tokens in parallel, making the process more efficient than the vanilla speculative decoding method, which predicts draft tokens sequentially.

Here's how it works more specifically:

- Initial Stage: Suppose the input token list is $\{x_1, x_2, \cdots, x_t\}$, we start with the original model's last hidden states (H_t) at position t.
- Draft Stage: We use the Otter-inserted H'_t to map to K new decoding heads. Each decoding head k predicts the next $k+1$ tokens in parallel. Therefore, we obtained draft tokens: $\{x_t, x_{t+1}, \cdots, x_{t+k+1}\}$
- Verification Stage: The new token list with the draft tokens: $\{x_1, x_2, \cdots, x_t, x_{t+1}, \cdots, x_{t+k+1}\}$ is then verified by the original LLM. Suppose the first m tokens are accepted, then we have the final current generation token list: $\{x_1, x_2, \cdots, x_t, x_{t+1}, \cdots, x_{t+m}\}$. This new list becomes the input in step 1 for the next iteration. Note that H'_{t+m} is already computed in Otter with the verification process.

In the vanilla speculative decoding setting, the inference process takes several draft model forward passes to draft and one base model forward pass for verification. However, in the Otter/Medusa setting, the inference process will only use the k extra heads to draft and then take one base model forward pass for verification. Draft tokens are predicted in parallel, and the draft heads have significantly fewer parameters than draft models. This makes Otter/Medusa faster and less computationally intensive.

B Task Heads in Otter

The final layer of the Otter-modified transformer outputs hidden state $\tilde{H} = [H_o : H']$, where H_o is the original hidden state. For classification-related tasks, such as the helpful and harmless alignment and detoxification experiments, we have the projection layer defined as follows:

$$\begin{aligned} o_{lm} &= lm\_head(H_o) \\ o_{otter} &= \sigma(W_{o_n} H') \end{aligned} \tag{7}$$

where $W_{o_n} \in \mathbb{R}^{1 \times h'}$ is trainable matrix that projects H' to a singular tensor. The original output

Methods	Num Params	Training Time
ARGS	6.74B	163mins
Otter	8.51B	108mins

Table 8: The training time of ARGS and Otter comparison on preference alignment task

o_{lm} is obtained via the original lm_head project layer.

For generation-related tasks, such as speculative decoding tasks, we define the projection layer as follows:

$$\begin{aligned} o_{lm} &= lm\_head(H_o) \\ H_m &= W_{h_m} H' \\ o_{otter} &= lm\_head(H_m + H_o) \end{aligned} \tag{8}$$

where $W_{h_m} \in \mathbb{R}^{h_o \times h'}$ is a trainable matrix to map H' to H_m that has the same hidden dimension as H_o.

C Code Modification

We demonstrate that the entire modeling architecture of LLMs remains unchanged, with the exception of a single-line modification for RMSNorm. The following code provides the necessary adjustment to RMSNorm in the PyTorch environment.

```
# Modified RMSNorm function.
def RMSNorm(hidden_states):
    # change from:
    # variance = hidden_states.pow(2).
        mean(-1, keepdim=True)
    # to:
    variance = hidden_states[...,
        :ori_hdim].pow(2).mean(-1,
        keepdim=True)

    # below lines are not changed
    hidden_states = hidden_states *
        torch.rsqrt(variance +
        variance_epsilon)
    return weight * hidden_states
```

D Training Time Analysis

In general, the training for Otter is faster than the baselines, as only a few parameters need to be trained. To illustrate this difference, we provide training time for the helpful and harmless human preference alignment task. All models were trained on the same 4 NVIDIA A100 80G GPUs with DeepSpeed zero-stage 2. The per-device batch size is 1 and the gradient accumulation step is 8.

E Additional Preference Alignment Results

Besides Llama-7b-chat already experimented in Table 1, here we supplemented the ARGS and Otter methods on the Llama2-7b-instruct model to evaluate their performance on the human preference alignment task. As shown in Table 9, the Llama2 model itself performs better than Llama in terms of generating more helpful and harmless responses (Llama-7b obtained 3.981 and 3.757 average rewards), both ARGS and Otter can further enhance the model's performance, leading to higher average reward. More importantly, Otter significantly reduced time, and space overhead while keeping the same performance as ARGS.

Method	Average Reward ↑	Diversity ↑	Coherence ↑	Time Overhead ↓	Space Overhead ↓
Greedy	4.468	0.583	0.454	1.0x	1.0x
Greedy+ARGS	**5.221**	0.617	0.461	2.06x	2.04x
Greedy+Task Head Only	4.536	0.542	0.497	1.01x	1.03x
Greedy+Otter	5.175	0.739	0.515	**1.03x**	**1.25x**
Top-k	4.112	0.695	0.479	1.0x	1.0x
Top-k+ARGS	4.819	0.702	0.486	2.16x	2.04x
Top-k+Task Head Only	4.360	0.689	0.472	1.02x	1.03x
Top-k+Otter	4.807	0.820	**0.528**	1.03x	1.25x
Top-p	4.146	0.624	0.417	1.0x	1.0x
Top-p+ARGS	4.876	0.669	0.445	2.18x	2.04x
Top-p+Task Head Only	4.297	0.628	0.458	1.02x	1.03x
Top-p+Otter	4.864	**0.783**	0.507	1.04x	1.25x

Table 9: Comparison of ARGS and Otter using Llama2-7b-chat