ARCHON: AN ARCHITECTURE SEARCH FRAMEWORK FOR INFERENCE-TIME TECHNIQUES

Anonymous authors
Paper under double-blind review

ABSTRACT

Inference-time techniques are emerging as highly effective tools to enhance large language model (LLM) capabilities. However, best practices for developing systems that combine these techniques remain underdeveloped due to our limited understanding of the utility of individual inference-time techniques and the interactions between them. Additionally, efficiently and automatically searching the space of model choices, inference-time techniques, and their compositions is challenging due to the large design space. To address these challenges, we introduce ARCHON, a modular framework for selecting, combining, and stacking layers of inference-time techniques to construct optimized LLM systems for target benchmarks. Rather than relying on a single LLM called once, we leverage a diverse set of LLMs and inference-time techniques, creating *LLM systems greater than the sum of their parts*. ARCHON defines an extensible design space, encompassing techniques such as generation ensembling, repeated sampling, ranking, fusion, critiquing, verification, and unit testing. It transforms the problem of building LLM systems into a hyperparameter optimization objective. Given the available LLMs, inference-time techniques, and compute budget, ARCHON utilizes hyperparameter search techniques to discover optimized architectures for target benchmark(s). We evaluate ARCHON architectures across a range of instruction-following, reasoning, and coding benchmarks, including MT-Bench, Arena-Hard-Auto, AlpacaEval 2.0, MixEval, MixEval Hard, MATH, and CodeContests. ARCHON architectures outperform frontier models, such as GPT-4o and Claude 3.5 Sonnet, on these benchmarks, achieving an average accuracy increase of 15.1 percentage points by using all available LLMs.

1 INTRODUCTION

Inference-time techniques are gaining traction as effective methods for improving model capabilities. Examples include generation ensembling, ranking, and fusion, where models in the ensemble are queried in parallel, their responses are ranked, and the best ones are fused into a single, higher quality output, respectively (Jiang et al., 2023b; Wang et al., 2024). Other types of inference-time techniques are based on querying a single LLM successively (via repeated sampling) and using a voting strategy or unit tests to select the top generation (Brown et al., 2024; Chen et al., 2024; Li et al., 2024a). We divide these existing inference-time techniques into three categories: *generative*, meaning that new candidate responses are drawn from the models (e.g. generation ensembling and repeated sampling), *reductive*, meaning that the existing responses are aggregated or filtered to keep the top responses (e.g. fusion and ranking), or *comparative*, meaning they provide analysis of candidate responses (e.g. critiquing and unit testing), as shown in Table 2.

Recent work has made progress towards building robust *inference-time architectures*, which are systems composed of one or more large language models (LLMs) and inference-time techniques. Examples include Mixture-of-Agents (MoA) (Wang et al., 2024) and LLM-Blender (Jiang et al., 2023b), as well as single-model systems like LeanStar (Lin et al., 2024) and rStar (Deng et al., 2024). However, our experiments show that existing architectures, such as MoA, still suffer from lack of generalization and become significantly less effective beyond the task(s) they were developed on (see Section 4.2). We argue that designing effective and generalizable inference-time architectures requires:

- **Understanding the Utilities of Inference-Time Techniques**: Inference-time architectures typically delegate their additional inference budget towards more model sampling calls (Chen et al., 2024; Brown et al., 2024), which can be effective for math and coding tasks. Other tasks such as instruction-following and reasoning are shown to benefit from additional techniques, including ranking and fusion (Wang

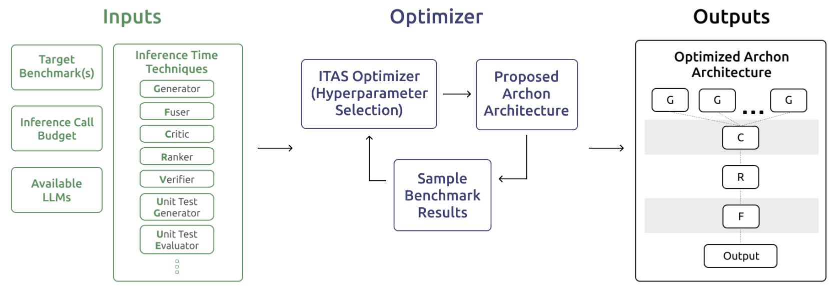

Figure 1: **Overview of ARCHON Framework**: Inference-Time Architecture Search (ITAS) requires the following inputs: target benchmarks, inference call budget, available LLMs, and available inference-time techniques (**left**). The ITAS algorithm uses Bayesian optimization (Snoek et al., 2012) (Section A.6) to select and test different ARCHON configurations (**middle**) before returning the optimized ARCHON architecture (**right**) for the target benchmarks (Section 3.3).

et al., 2024; Jiang et al., 2023b). While all of these methods are valuable, *it is essential to identify which inference-time techniques are most effective for different task categories.*

- **Understanding the Interactions Between Inference-Time Techniques**: While previous studies analyzed these techniques individually (e.g. generation sampling in Chen et al. (2024)), *we need a more comprehensive understanding of the relationships between different inference-time techniques* across different tasks (e.g. is it better to use more models or generate more samples per model?).
- **Efficiently and Automatically Searching the Large Design Space of Inference-Time Architectures**: Given a set of available LLMs and target tasks, there is currently no single prevailing inference-time architecture for maximizing downstream accuracy across all tasks (Table 1). The search space of inference-time architectures is expansive, requiring practitioners to make several key configuration decisions: *which LLMs to use, how many times to sample them, how to combine the candidate generations, what inference-time techniques to perform on the candidates, and more*. These motivate the need for adaptive and automated architecture search approaches.

In our work, we address each of these challenges. Firstly, we **evaluate the utilities of a comprehensive set of existing and proposed inference-time techniques** across instruction-following, reasoning, and coding tasks. Using both open-source and closed-source models, we examine a range of techniques such as *ensembling, fusion, ranking, critiquing, and verification* and introduce new methods such as *model-based unit test generation and evaluation* (Sections 3.1 and 3.2).

Secondly, we **analyze the interactions between inference-time techniques**, and explore the benefits of adding new models and new techniques individually. We find that candidate fusion substantially improves the quality of the final response generation, and when combined with additional techniques like critiquing, verifying, and ranking, can improve generation quality beyond the oracle best candidate from individual (non-fused) responses (Figure 3; Figure 7). Additionally, we find that candidate verification, unit test generation, and unit test evaluation are most effective for reasoning tasks, whereas critiquing and ranking are effective across instruction-following and reasoning tasks (Section 3.1; Table 12).

Thirdly, drawing upon our analysis of inference-time techniques, we present **ARCHON**, a framework for building inference-time architectures. ARCHON utilizes automatic **inference-time architecture search (ITAS)** algorithms to maximize generation quality for a wide range of tasks, including instruction-following, reasoning, and coding. Our ARCHON framework and ITAS algorithms draw inspiration from neural architectures and neural architecture search (NAS) (Zoph & Le, 2017; Ren et al., 2021; Liu et al., 2018; 2021), respectively. ARCHON is constructed of *layers of LLMs*, in which LLMs within the same layer run in parallel, but each layer runs sequentially. The layers perform different inference-time techniques, either transforming the number of candidate responses through generation and fusion (analogous to linear transformations) or reducing the number of candidate responses to improve quality (akin to non-linearities) (Section 3.1). The number of generators, samples per model, fusion layers, fusion models per layer, and more, are all treated as hyperparameters for optimization in our ITAS algorithms (Section 3.3).

Overall, our work makes the following contributions: **(1)** We develop ARCHON, an open-source modular framework for designing LLM systems that combine inference-time techniques (Section 3.1). We utilize ITAS as the optimizer engine for ARCHON, which enables automated inference-time architecture search for target benchmarks, leveraging Bayesian optimization (Snoek et al., 2012; Nardi et al., 2019) (Section 3.3). ARCHON is plug-and-play, allowing users to select from existing inference-time techniques (or add new ones) and specify their desired objective functions to optimize for accuracy, latency, and cost. **(2)** We demonstrate increased performance as we scale up the layers of inference-time techniques and combine multiple approaches together, allowing us to discover effective new combinations of inference-time techniques (Sections 3.2, 4.2, A.2). We find that sequentially applying critique, ranking, top-k selection, and then fusion is a highly effective composition (Figure 3; Table 1), and we demonstrate the effectiveness of model-based unit test generation and evaluation for improving coding capability (Figure 5). **(3)** Our best ARCHON architectures surpass both single-call LLMs (e.g. GPT-4o and Claude-3.5 Sonnet) and prior top-performing inference-time architectures (e.g. Mixture-of-Agents (Wang et al., 2024)), boosting state-of-the-art performance by 15.1 percentage points, on average, across a diverse set of instruction-following, reasoning, and coding benchmarks (Table 1; Figure 5): MT-Bench, Arena-Hard-Auto, Alpaca-2.0 Eval, MixEval, MixEval Hard, MATH, and CodeContests (Zheng et al., 2023; Li et al., 2024b; 2023; Ni et al., 2024; Hendrycks et al., 2021; Li et al., 2022). Even when just using open-source LLMs, ARCHON architectures on average surpass single-call state-of-the-art (SOTA) LLMs by 11.2 percentage points.

2 RELATED WORK

Scaling Laws of Language Models: Language models (Touvron et al., 2023; Jiang et al., 2023a; Team et al., 2024a; OpenAI et al., 2024) have transformed the field of artificial intelligence across a vast number of domains and tasks. LLMs are pretrained on substantial amounts of textual data before being further aligned with human preferences through instruction fine-tuning (Wei et al., 2022; Chung et al., 2022), DPO (Rafailov et al., 2023), KTO (Ethayarajh et al., 2024), RLAIF (Bai et al., 2022b), and other techniques. As language models continue to gain improved abilities with further scaling of data, parameters, and compute (Kaplan et al., 2020; Gadre et al., 2024), the cost of developing new LLMs is ever increasing, requiring the curation of trillions of new tokens as well as substantial GPU-hours for pretraining. Furthermore, as the current state-of-the-art in LLMs are primarily closed-source APIs, such as OpenAI's GPT-4o (OpenAI et al., 2024), Google's Gemini (Team et al., 2024b) and Anthropic's Claude (Anthropic, 2024), it is difficult to effectively explore and push the frontier of existing LLMs without being able to manipulate the parameters of these closed-source models and employing techniques such as continual pretraining (Jin et al., 2021), instruction fine-tuning (Wei et al., 2022), data mixing (Ye et al., 2024), chain-of-thought (Wei et al., 2023), among others.

Inference-Time Techniques: Inference-time architectures combine multiple frozen LLMs and inference-time techniques (e.g., generation ensembling, sampling, ranking, and fusion), achieving superior performance compared to individual models. Notable works include Mixture-of-Agents (MoA) (Wang et al., 2024), LLM Blender (Jiang et al., 2023b), RouteLM (Ong et al., 2024), Smoothie (Guha et al.), and various approaches around *compound AI*, which are AI systems that use multiple components (e.g. LLMs, retrievers, tool use, APIs, etc.) (Chen et al., 2024; Davis et al., 2024; Lewis et al., 2020; Shao et al., 2024; Kapoor et al., 2024). LM frameworks like DSPy (Khattab et al., 2023) and TextGrad (Yuksekgonul et al., 2024) have emerged for orchestrating LMs and other components. Even with a single LLM, various techniques can improve performance by building better reasoning strategies, such as OpenAI's o1 (OpenAI, 2024b), Chain of Thought (Wei et al., 2023), and Branch-Solve-Merge (Saha et al., 2024), as well as inference-time frameworks, such as ADAS (Hu et al., 2024) and AFlow (Zhang et al., 2024).

Despite these advancements, challenges remain in developing inference-time architectures. Many architectures focus on additional generations (Jiang et al., 2023b; Chen et al., 2024; Davis et al., 2024), which is effective for reasoning tasks (Brown et al., 2024). However, for tasks like chat and instruction-following, techniques such as fusion and ranking are useful (Wang et al., 2024; Jiang et al., 2023b). For tasks without built-in verification, additional compute for reasoning and verification can improve accuracy (Davis et al., 2024). We still lack understanding of trade-offs between different inference-time techniques. Prior studies have explored limited aspects of configurations, often focusing on specific benchmarks (Jiang et al., 2023b; Wang et al., 2024; Chen et al., 2024; Li et al., 2024a). It's crucial to efficiently develop inference-time architectures, as optimal configurations vary based on benchmarks, available models, and inference call limits (Section 4.2). To address these challenges, we analyzed multiple inference-time techniques (Section 3.1) and developed the ARCHON framework for automating the development of inference-time architectures with ITAS (Section 3.3).

3 INFERENCE-TIME TECHNIQUES FOR ARCHON

With the proliferation of inference-time techniques, ARCHON introduces a simple framework that unifies different approaches, providing structure for understanding and combining various techniques. Our framework not only incorporates methods for generating, ranking, and fusing candidates inspired by previous work

(Wang et al., 2024; Jiang et al., 2023b) but also integrates new approaches for critiquing, verifying, and unit testing candidate responses.

Below, we elaborate on the structure, inputs, and outputs of each of the inference-time techniques, which we also include in Table 2. Then, we discuss how to combine the different techniques into an inference-time architecture (Section 3.2) and the relationships between the different inference-time techniques (Section A.2) before finally exploring automatic approaches for constructing inference-time architectures (Section 3.3).

3.1 LLM Components of Archon

In this section, we discuss the *LLM components* of Archon, which are LLMs that perform a specific inference-time technique. We test an array of different components inspired by recent work, incorporating approaches for generating, ranking, and fusing candidates (Wang et al., 2024; Jiang et al., 2023b) as well as approaches for improving candidate response quality through critiquing, verifying, and unit testing (Bai et al., 2022a; Zheng et al., 2023). The components and their prompts are summarized in Table 2 and Section A.1.

Generator is an LLM that takes in the instruction prompt and outputs candidate responses. Generators can be called in parallel to perform *generation ensembling* (i.e. calling multiple LLMs in parallel) (Wang et al., 2024), or sampled multiple times (Brown et al., 2024). When calling the Generators in parallel, you can sample one or more LLMs one or more times. The number of models, samples, and temperature for generation can be varied based on model configuration.

Fuser is an LLM that, given an instruction prompt and a set of proposed responses as input, combines these responses to generate one or more higher-quality fused responses that better address the instruction prompt.

Ranker is a LLM that, given an instruction prompt and a set of proposed responses as input, ranks the candidate generations based on their quality, producing a ranked list of responses as output.

Critic is an LLM that, given an instruction prompt and a set of proposed responses as input, produces a list of strengths and weaknesses for each response, which is then used to improve the quality of the final response (Section 3.2; Figure 3).

Verifier is a LLM that verifies whether a provided candidate response has appropriate reasoning for a given instruction prompt. It proceeds in two stages: **Stage #1** takes in the instruction prompt and a candidate response as input and outputs reasoning for why the candidate response is correct; **Stage #2** takes in the instruction prompt, candidate response, and produced reasoning before outputting reasoning and a verdict (i.e. binary [Correct] or [Incorrect]) for whether or not the candidate response is correct according to the provided instruction prompt and reasoning.

Unit Test Generator is a LLM that takes only the instruction prompt as input and outputs a list of unit tests that assess the accuracy and relevance of candidate responses. These unit tests are verified by the Unit Test Evaluator to rank different responses. Each test is a concise statement that can be passed or failed. We make the number of unit tests generated a configurable choice for the unit test generator but we find 5-10 generated unit tests to be most effective with our set of LM prompts (Section 4.2; Figure 5). For examples, please see Table 10.

Unit Test Evaluator is a LLM that takes in the instruction prompt, candidate response(s), and set of unit tests before outputting the candidate response(s), ranked in descending order by how many unit tests they pass. We use model-based unit test evaluation by prompting the LLM to provide reasoning and verdicts for each unit test across each of the candidate responses. By aggregating the unit test verdicts for each candidate response, the unit test evaluator ranks the candidate responses. For reasoning tasks, particularly coding tasks, it can be useful to compare different candidate responses by the number of unit tests they pass to gauge for quality (Figure 5).

3.2 Combining the LLM Components

Overview: Inspired by the structure of neural networks (Hinton et al., 1992), Archon is constructed of layers of LLM components (Figure 1; Section 3.1). Each layer is composed of sets of these LLM components that are called in parallel, performing a text-to-text operation to the instruction prompt and the candidate responses from the previous layer. Furthermore, like a neural network, some layers perform *transformations* of the provided list of strings (e.g. the Generator and Fuser components), converting a list of strings into a different list of strings (the numbers of candidates can vary from the original number of candidates). Other components introduce non-linearities into the Archon structure, performing filtering of the list of strings (e.g. Ranker and Verifier). Ultimately, the inputs and outputs for each layer is always a list of strings, whether that is the instruction prompt (e.g. a list with a single string) or a list of candidate responses (e.g. a list of many strings). If a list of strings are outputted at the last layer of the ARCHON structure, the first string in the list is returned.

Unlike a classical neural network, no weights are learned between the LLM components and the layers; in turn the ARCHON architecture can be deployed off-the-shelf without any tuning. Additionally, a single state is transformed sequentially from the input layer to the final output; this single state is the initial instruction prompt and the current candidate responses. In Figure 2, we provide an example ARCHON architecture composed of six layers.

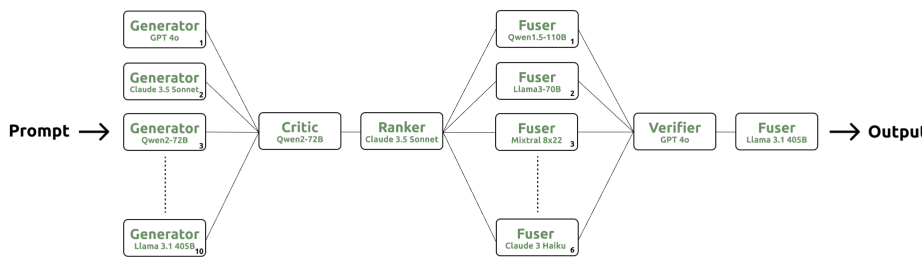

Figure 2: **Example ARCHON Architecture**: This architecture starts with ten generator models, followed by a critic model, a ranker model, one layer of six fuser models, a verifier model, and finishes with a fuser model.

Rules for Construction: The LLM components in Section 3.1 can only be placed in specific orders:

1. Only one type of module can be present in any given layer.
2. Generator components must and can only be placed in the first layer of ARCHON; you can put multiple Generators or a single Generator in the layer.
3. The Critic component must come before a Ranker or a Fuser, otherwise the generated strengths and weaknesses cannot be incorporated into generation ranking or fusion, respectively.
4. Ranker, Critic, Verifier, and Unit Test Generator/Evaluator layers can go anywhere in the ARCHON structure (besides the first layer); for each of these components, it must be the one and only module in its layer.
5. Fuser components can also be placed anywhere in the ARCHON structure (besides the first layer); you can put multiple Fusers or a single Fuser in the layer.
6. Unit Test Generators and Evaluators are placed in layers next to each other: generator first, then evaluator.

We provide an overview of the available placements and configurations for each LLM module in Table 3. We also analyze the different interactions between each LLM component and find increased ARCHON performance as we scale the "layers" of inference-time techniques by combining multiple approaches together sequentially (Section A.2).

Performance Gains from Scaling Inference-Time Techniques: By scaling both the layers of inference-time techniques and the diversity of inference-time techniques included, we were able to significantly improve ARCHON performance across instruction-following, reasoning, and coding tasks (Figure 3). In particular, repeated model sampling and additional ensemble models led to substantial gains (Figure 4), leading to 9.3 and 18.5 percentage point increases, respectively. On coding tasks, additional samples provided the largest marginal benefits, leading to a 56% boost in Pass@1 for CodeContests when repeated sampling was combined with model-based unit test generation/evaluation (Figure 1). Multiple layers of Fusers were found to be particularly effective for instruction-following tasks, delivering notable performance improvements as the number of layers increased (Figure 3). In reasoning tasks, incorporating the Verifier and Unit Test Generator/Evaluator modules alongside the Fuser improved performance by filtering out flawed responses, contributing to significant performance gains in tasks like MixEval and CodeContests. For detailed analysis of interactions between LLM components, please see Section A.2, where we perform a series of ablation experiments in which we vary ARCHON component combinations (Table 12) and the models used in the combinations (Table 13; Table 14; Table 15; Table 16).

3.3 INFERENCE-TIME ARCHITECTURE SEARCH (ITAS)

In this section, we explore different approaches for finding the best inference-time architecture (for a given task) through *inference-time architecture search* (ITAS). Due to compute resources, we pre-filtered certain

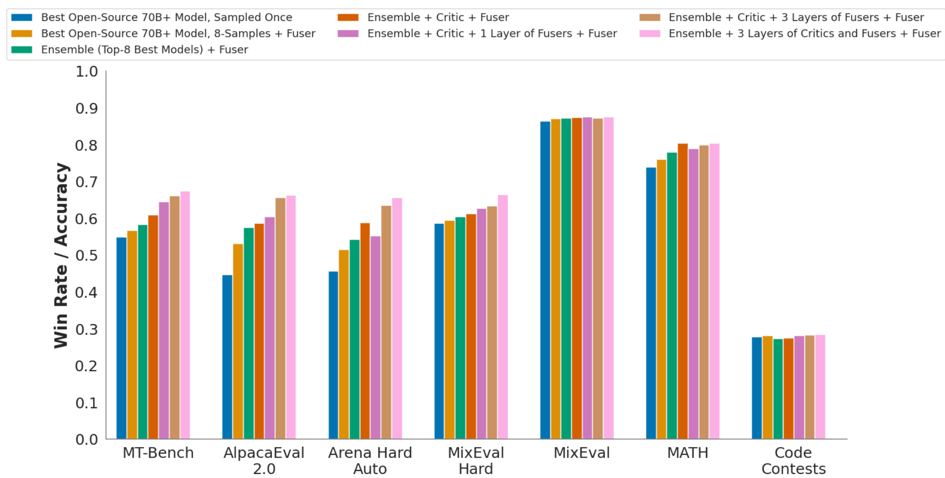

Figure 3: **Performance Gains from Scaling *Layers* of Inference-Time Techniques**: We generally observe performance improvements as we scale the critic and fusion layers. Compared to sampling the best open-source model once, our inference-time architecture with an 8-model ensemble, 3 layers of critic and fusion (8 models in each layer), and a final fusion performs on average 17.3% higher. For MixEval and CodeContests, we find that alternative inference-time architectures are more effective than generator ensembles and fusion layers. We break-down our results for MixEval and MixEval-Hard by subdataset in Section 4.2 (Table 31; Table 32). For CodeContests, we show the effectiveness of increased generator sampling combined with model-based unit test generation/evaluation in Figure 5. The results were calculated from 10 independent evaluation runs.

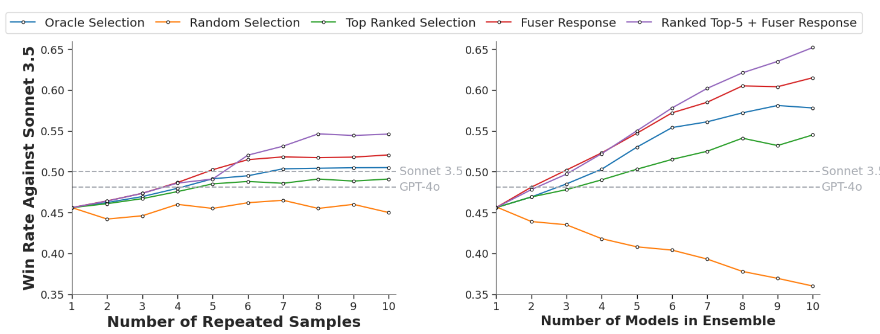

Figure 4: **Performance Gains from Repeated Sampling, Ensembling, Ranking, and Fusing on Arena-Hard-Auto**: The ARCHON win-rate continues to grow significantly as we scale model sampling **(left)** or add additional models to the generator ensemble **(right)**, increasing by 9.3% and 18.5%, respectively. These best results are achieved by selecting the top-5 responses and fusing them. The ensemble models are added based on their individual performance on this task, from best to worse (Table 18). The results were calculated from 10 independent evaluation runs.

ways of combining LLM components to reduce the search space while still building effective inference-time architectures. While it is possible to expand the search space of potential ARCHON architectures (e.g. different temperatures for generative LLM components, alternative prompts for each LLM component, multiple layers of Generator modules, additional LLM components for ARCHON, etc.), we use our analysis from Section 3.2 to selectively limit our search space to configurations that fit our rules for ARCHON: starts with a layer of Generator modules, followed by layers performing fusing, ranking, critiquing, verifying, and unit testing.

Search Hyperparameters: We selected five main axes for the hyperparameters in our search:

1. **Top-K Generators for Ensemble**: The top-K models to be used for the initial Generator ensemble, ranges from 1 to 10. The top-K models are the best-K LLMs for the given task, based on their individual performances (Table 18).

2. **Top-K Generator Samples**: The number of samples gathered from each Generator in the ensemble (it is the same for all the models), ranges from 1 to 5. For Code-Contests, we explore high sample settings: [1, 10, 100, 500, 1000].
3. **Number of Fusion Layers**: Ranges from 1 to 4. The last fusion layer will always have a single Fuser.
4. **Top-K Fusers**: Number of models used for each fusion layer, ranges from 2 to 10 and increases by 2 each time.
5. **Evaluation Layer**: Option to add a Verifier, Unit Test Generator/Evaluator, or neither before the last Fuser layer.

By combining all the hyperparameters, we create a search space of 18,750 configurations by multiplying each of the configuration option counts together ($10 * 5 * 5^{(4-1)} * 3 = 18,750$). However, we remove configurations that are not viable: configurations in which the number of initial generations exceeds the context window of the fusers (i.e. 24 candidate generations) and configurations with only one fuser layer but multiple fusers declared. This reduces our search space to 9,576 configurations. For these configurations, we add critic and ranker layers before each fuser layer since they've been shown to have added benefits across the benchmarks explored (Figure 7; Figure 3). The ranker selects the top-5 candidate generations to send to the next layer. The unit test generator uses a default setting of 5 unit tests generated.

Search Methodology: Within ITAS, we use *Bayesian Optimization* to select the most promising hyperparameter configurations (Snoek et al., 2012; Nardi et al., 2019). For generator ensemble, we add the models to the pool in a greedy manner, starting from the best performing model (on average) on the target benchmarks. For each fuser ensemble layer, we use the same approach, adding the best fuser models in a greedy manner. To rank them, we evaluate their fusion performance on the samples from an ensemble of top 10 generator models. We found that the best generator and fusion models could vary widely across datasets, making it beneficial to perform these rankings for new datasets (Table 18). For search, we use a 20% sample of each dataset for guiding architecture search to improve the evaluation speed while getting meaningful development signal.

Overall, Bayesian Optimization was the most effective search algorithm for constructing ARCHON systems, outperforming other methods like random and greedy search by more efficiently finding optimal configurations (Section A.6). It found the best architectures in 96.0% of iterations and required 88.5% fewer evaluations than greedy search and 90.4% fewer than random search (Figure 13). The effectiveness of Bayesian optimization increases with the number of initial testing points, up to around 230-240 samples, after which further testing is better focused on configuration search (Table 26). However, for limited inference call budgets (<20 calls), Bayesian optimization is less effective, and traditional methods like greedy search may perform comparably (Table 27).

For our implementation, we use a Python package of Bayes global optimization with Gaussian processes. As inputs, our Bayes implementation takes in the integer lists of configuration choices for the generators (i.e. number of models and samples), layers of fusers, numbers of fusers per layer, and final verifier / unit tester. Bayes algorithm then proceeds to select different combinations of integers from these lists in its search process, iteratively evaluating each generated ARCHON architecture on the development set to find the optimal ARCHON configuration. For more information, please see Section A.6.

4 EXPERIMENTS

Our experiments focus on four questions: **(1)** how does ARCHON compare to existing SOTA LLMs and multi-LLM systems? **(2)** how does ARCHON performance compare across tasks? **(3)** how does ARCHON performance compare when optimized for a set of tasks vs. an individual task? **(4)** what are ARCHON's current limitations and plans for future work?

4.1 BENCHMARKS AND MODELS

Benchmarks: We evaluate our models with several benchmarks for instruction-following, reasoning, and coding: MT-Bench (Zheng et al., 2023), AlpacaEval 2.0 (Li et al., 2023), Arena Hard Auto (Li et al., 2024b), MixEval (Ni et al., 2024), MixEval-Hard, MATH (Hendrycks et al., 2021), and CodeContests (Li et al., 2022). We provide an overview of each dataset in Table 29, where we compare their query counts, scoring type, evaluation metrics, reference models, and judge models. Since we perform ITAS on a randomly sampled 20% subset of each benchmark, we evaluate on the remaining held-out 80% subset of the benchmark (Table 1; Figure 5) (for ARCHON performances on the entire benchmarks, please see Table 28). The delta between the ARCHON performance on the entire benchmark vs. 80% held-out subset is relatively small:

							MT Bench	Alpaca Eval 2.0	Arena Hard Auto	MixEval Hard	MixEval	MATH
		Approaches	Infer. Calls	Input Tokens	Output Tokens	TFLOPs per Token	W.R.	L.C. W.R.	W.R	Acc.	Acc.	Pass @1
Baselines	LM	GPT-4o	1	95	549	Unk.	44.2% ±0.5	57.8% ±0.6	80.6% ±0.6	63.4% ±0.2	87.5% ±0.3	73.2% ±0.4
		Claude 3.5 Sonnet	1	105	602	Unk.	N/A	52.7% ±0.4	81.4% ±0.4	68.7% ±0.2	89.1% ±0.2	73.1% ±0.7
		Llama 3.1 405B	1	118	631	0.81	44.1% ±0.3	40.7% ±0.5	64.5% ±0.7	66.0% ±0.3	88.2% ±0.2	75.2% ±0.5
	LM Systems	MoA	19	25,109	17,422	1.36	51.6% ±0.6	65.4% ±0.3	84.5% ±0.3	62.3% ±0.4	86.9% ±0.2	73.9% ±0.6
		MoA Lite	7	7,943	6,437	0.52	45.6% ±0.4	59.6% ±0.7	88.3% ±0.5	60.9% ±0.3	86.4% ±0.3	71.8% ±0.3
		ADAS	52	72,804	44,872	Unk.	66.3% ±0.7	60.1% ±0.5	85.4% ±0.4	64.2% ±0.2	87.0% ±0.2	74.5% ±0.8
		AFlow	48	68,596	41,748	Unk.	62.4% ±0.2	57.8% ±0.6	83.2% ±0.6	63.5% ±0.3	87.2% ±0.4	73.2% ±0.2
		O1 Mini	Unk.	112	Unk.	Unk.	57.1% ±0.3	57.8% ±0.4	79.3% ±0.8	70.8% ±0.2	87.0% ±0.3	81.7% ±0.4
		O1 Preview	Unk.	112	Unk.	Unk.	56.3% ±0.5	59.3% ±0.5	81.7% ±0.3	72.0% ±0.4	87.5% ±0.2	73.5% ±0.5
Archon	Open Src.	General Purpose	35	51,113	31,508	3.14	67.2% ±0.4	63.3% ±0.6	85.6% ±0.5	65.3% ±0.3	86.2% ±0.2	76.6% ±0.6
		Task Specific	44	63,157	39,949	3.71	71.1% ±0.6	67.1% ±0.4	89.6% ±0.4	67.5% ±0.2	88.8% ±0.3	81.9% ±0.3
	Closed Src.	General Purpose	32	52,747	27,894	Unk.	72.7% ±0.3	63.9% ±0.7	86.2% ±0.7	67.5% ±0.4	87.2% ±0.2	77.9% ±0.7
		Task Specific	40	59,085	37,271	Unk.	77.0% ±0.5	68.9% ±0.5	90.5% ±0.3	72.6% ±0.3	89.5% ±0.3	81.6% ±0.4
	All Src.	General Purpose	35	50,427	30,461	Unk.	76.2% ±0.7	66.4% ±0.3	89.8% ±0.6	69.8% ±0.2	87.3% ±0.4	79.3% ±0.5
		Task Specific	39	58,250	36,114	Unk.	**79.5% ±0.4**	**69.0% ±0.6**	**92.5% ±0.5**	**72.7% ±0.3**	**89.7% ±0.2**	**82.1% ±0.6**

Table 1: ARCHON*'s Strong Performance with ITAS Optimization on Open Source, Closed Source, and All Source Models*: Consistent outperformance over state-of-the-art LLMs across explored benchmarks. The standard error numbers were calculated from 10 independent evaluation runs.

only 0.44 percentage points, on average, across these datasets with an S.D. of 0.20 percentage points. For MixEval and MixEval Hard, we use the 2024-06-01 dataset release. For MT Bench, AlpacaEval 2.0, and Arena-Hard-Auto, the reference models are Claude 3.5 Sonnet, GPT-4-Turbo, and GPT-4-Turbo, respectively, while the judge models are GPT-4-0314, GPT-4-Turbo, and GPT-4-Turbo, respectively. For MATH, we evaluate a random sample of 200 problems from the dataset's test set. For CodeContests, we evaluate on the 140 test set questions that do not include image tags in the problem description.

Models: We test the efficacy of the ARCHON framework by creating various different ARCHON architectures (Section 4.4) across three model categories: 8B or less parameter models, 70B or more parameter models, and closed-source model APIs. For our 8B and 70B+ models, we selected the top-10 performing chat models for each parameter range on the Chatbot Arena Leaderboard (Chiang et al., 2024) as of July 2024. For our ARCHON architectures, we explore multiple model types: open-source, closed-source, and *all-source* (i.e. both open-source and closed-source available). For our closed-source model APIs, we include GPT-4o, GPT-4-Turbo, Claude Opus 3.0, Claude Haiku 3.0, and Claude Sonnet 3.5. We list and compare all of the models tested in the ARCHON framework in Table 17 and Table 18. For all the LLMs utilized and every ARCHON component, we set the generation temperature to 0.7. As baselines, we compare ARCHON against both SOTA LLMs (GPT-4o (OpenAI et al., 2024), Claude 3.5 Sonnet (Anthropic, 2024), and Llama 3.1 405B Instruct (AI@Meta, 2024)) as well as SOTA inference-time architectures (OpenAI's O1 (OpenAI, 2024a), MoA (Wang et al., 2024), ADAS (Hu et al., 2024), and AFlow (Zhang et al., 2024)).

4.2 ARCHON VS. CLOSED-SOURCE LLMS AND OTHER INFERENCE-TIME ARCHITECTURES

We start by comparing ARCHON architectures to existing SOTA closed-source LLMs and inference-time architectures across a set of instruction-following, reasoning, and coding tasks. Based on our results in Table 1, we find that ARCHON architectures consistently match or surpass existing approaches across all the benchmarks explored. ARCHON architectures with open-source models demonstrate a 11.2% average improvement over SOTA open-source approaches; for its worst performance, our open-source ARCHON architectures are only 3.6% above SOTA open-source approaches on AlpacaEval 2.0. ARCHON architectures with closed-source models achieve SOTA performance across MT Bench, Arena-Hard-Auto, MixEval, and MixEval-Hard, leading to a 15.8% average improvement over closed-source LMs and a 6.8% average improvement over open-source inference-time frameworks (i.e. MoA, ADAS, and AFlow). Furthermore, compared to these open-source inference-time frameworks, Archon is 20% more inference call efficient while having higher performances on all benchmarks tested. We also find that our best Archon architectures use 15.1% less input tokens and 13.5% less output tokens compared to the best alternative open-source inference-time frameworks. Compared to O1-preview and O1-mini, ARCHON's best targeted architectures beat them by 8.1% and 9.7%, on average, on MT Bench, AlpacaEval 2.0, Arena Hard Auto, MixEval, MixEval Hard, and MATH. On CodeContests,

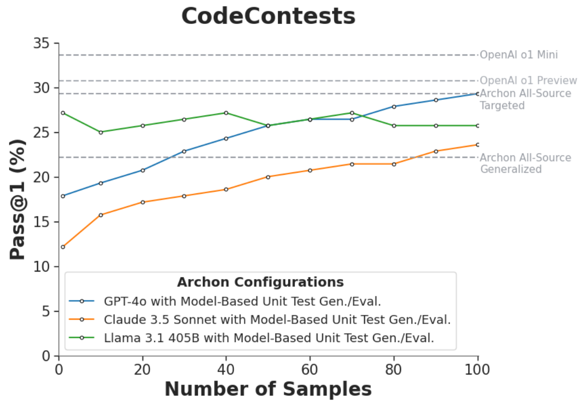

Figure 5: ARCHON *Performance Gains from Combining Multi-Sampling with LLM-based Unit-test Generation/Evaluation*: Strong performance improvements in Pass@1 as we scale the number of samples for GPT-4o and Claude 3.5 Sonnet. The standard error numbers were calculated from 10 independent evaluation runs.

O1-preview and O1-mini narrowly beats ARCHON by 1.7% and 5.3%, on average, as the O1 system is specially trained towards handling complex reasoning tasks like math and coding. Lastly, for approaches that use all models available, both open and closed-source, ARCHON achieves an average 10.9% improvement over existing SOTA single-call LLMs and an average 8.6% improvement over existing inference-time frameworks.

4.3 ARCHON BY TASK

We analyze ARCHON performance by task style: instruction-following tasks that use pairwise ranking for scoring, reasoning tasks that use accuracy-based metrics for scoring, and coding tasks that use Pass@1. On instruction-following tasks like MT Bench, AlpacaEval 2.0, and Arena-Hard-Auto, open-source ARCHON architectures outperform current open-source baselines by 10.0 percentage points, on average, while closed-source ARCHON outperforms current closed-source baselines by 20.1 percentage points (Table 1). On reasoning tasks like MixEval, MixEval-Hard, and MATH, open-source ARCHON architectures outperform existing open-source baselines by 2.9 percentage points while closed-source ARCHON architectures outperform current closed-baselines by 4.2 percentage points (Table 1). On coding tasks (i.e. CodeContests), open-source ARCHON architectures match existing open-source baselines (0.2 percentage points difference) and all-source ARCHON architectures outperform all-source baselines by 2.5 percentage points (Figure 5). All-source architectures of ARCHON outperform existing all-source baselines by 16.1 and 3.8 percentage points, on average, for instruction-following tasks and for reasoning tasks, respectively (Table 1).

Instruction-Following and Reasoning: With ARCHON, multiple models used for Generators and the depth of fusion layers lead to performance boosts on instruction-following tasks, increasing the richness of responses and allowing multiple iterations for step-by-step instruction-following (Table 19). For reasoning, while the performance boost from ARCHON is smaller when we consider the *aggregate* scores for MixEval and MixEval-Hard, we do see meaningful increases in performance when we create inference-time architectures for each individual task under MixEval and MixEval-Hard (Table 31; Table 32). When we create individual ARCHON architectures for each subtask, we see 3.7 and 8.9 percentage point increases in accuracy, on average, for MixEval and MixEval-Hard, respectively. This finding suggests that reasoning tasks (e.g. math, sciences, logic) require more individualized inference-time architectures for their particular queries.

Coding: We have observed that ensembling, fusion, and ranking techniques have limited impact on CodeContests (Figure 3). For example, when we apply the general all-source architecture from Table 29 to CodeContests problems, we achieve small gains from ARCHON (see Figure 5). One contributing factor is that, unlike the distribution of instruction-following/reasoning tasks, coding tasks tend to have one or two LLMs that perform substantially better than the rest of models (Table 18). However, when we add unit test generation/evaluation, and scale the number of samples, ARCHON's performance on CodeContests improves significantly (Figure 5), allowing us to boost GPT-4o Pass@1 performance by 56% for Pass@1 (from 25 to 41 out of 140 questions). For model-based unit test generation/evaluation, we generate 5 unit tests and use the LM to evaluate each candidate response against the generated unit tests, allowing us to rank the different candidate responses (details are provided in Section A.1)

4.4 TASK-SPECIFIC AND GENERAL-PURPOSE ARCHON ARCHITECTURES

Task-Specific vs. General-Purpose: We also compare custom ARCHON architectures, specifically configured to a single evaluation dataset ("Task-specific ARCHON Architectures"), and a generalized ARCHON architecture configured to handle all the evaluation datasets ("General-purpose ARCHON Architectures") (Table 1). For our three model selection settings for ARCHON (i.e. open-source, closed-source, and all-source), we utilize ITAS to find targeted ARCHON architectures for each task (7 architectures total) and find a single generalized ARCHON architecture for maximizing performance over all the tasks (Table 1). The benchmarks are concatenated together and shuffled for generalized Archon architecture search. For examples of targeted and generalized ARCHON architectures, please see Figure 2 and Section A.4.

We utilize ITAS to find the generalized ARCHON architectures in Table 1 (Section 3.3), maximizing performance over all of the benchmarks explored except CodeContests. While we use ITAS to find a targeted ARCHON architecture for CodeContests, we exclude the dataset from the generalized ARCHON architecture search since we found that ARCHON architectures for coding tasks are most effective with a different set of inference-time techniques compared to instruction-following and reasoning tasks (i.e. increased model sampling combined with model-based unit test generation/evaluation) (Section 3.2; Figure 3). For open-source models, we find that our generalized ARCHON architecture only lags behind the specialized ARCHON architectures by 3.4 percentage points, on average, across all the benchmarks, demonstrating the robustness of the ARCHON architecture found by the ITAS algorithms (Table 1). We see similar gaps between the generalized and specialized ARCHON architectures for closed-source models (4.0 percentage points) as well as the all-source models (3.3 percentage points) (Table 1).

Insights from Architecture Construction: We include examples of our learned effective generalized ARCHON architectures constructed by ITAS in Section A.4. For instruction-following and reasoning tasks, we found a generalizable ARCHON architecture to be most effective with multiple layers of critic-ranker-fuser, chained sequentially to improve candidate generation (Figure 9). However, the specific models chosen for these LLM components could change task by task, with some tasks benefiting from using a single SOTA closed-source LLM for all the components (e.g. Arena-Hard-Auto and MixEval) (Figure 11) whereas others benefited from a diversity of LLMs in their ensemble (e.g. MT Bench and MixEval-Hard) (Figure 9; Figure 10). Regardless of models used, we found that scaling inference layers including critics, rankers, and fusers improved performance on instruction-following and reasoning tasks (Figure 3; Section A.4). For instruction-following and reasoning tasks, the verifier module is more effective than the unit test generation/evaluation module for task-specific ARCHON architectures (Section 3.2; Table 12). For coding tasks, we found a high-sample setting to be the most effective, with added layers of unit test generation and evaluation to boost the quality of the final candidate generation (Figure 12; Figure 5).

4.5 LIMITATIONS AND FUTURE WORK OF ARCHON

Parameter Count: The ARCHON framework is most effective with LLM with about 70B parameters or more. When we utilize the ARCHON architecture with only 7B open-source models, we get a notable decrease in performance (Table 21). The best 7B ARCHON configurations lag behind single SOTA (and much larger) models by 15.7% on across all the benchmarks, on average; 7B models work well for ranking but are less effective for critic and fusion.

Latency and Costs: Since ARCHON architectures make multiple LLM API calls successively for different operations (e.g. ensembling, critiquing, ranking, etc.), it can often take 5x more time than a single LLM API call (Section A.4). Furthermore, it can require calling multiple API endpoints for a single query, leading to increased expenditures (Table 22; Table 23). Note that these increases in compute costs and latency translate to higher quality responses, and can be justified in many application domains, such as science, math, programming, and complex customer service issues. For tasks in which speed is most preferred, future work should explore how distillation strategies (Sreenivas et al., 2024) could be used to pack the aggregate knowledge of ARCHON architectures into a smaller LM.

ARCHON Components: While ARCHON is a modular framework, allowing the easy incorporation of new LLMs, new inference-time techniques, and even tool use, we only explore seven LLM inference time techniques in our work (Section 3.1). The addition of new techniques is a promising avenue for future research. Furthermore, while different queries can be best suited by different ARCHON architectures (Table 31; Table 32), the ITAS algorithm selects the best single architecture for the evaluation set queries combined. Future architecture search could focus on dynamic selection of ARCHON components, LLMs, and tools on a query-by-query basis.

5 Reproducibility Statement

For the Archon model and benchmark configurations, we included the related information in Sections 4.1, 4.2, and A.1. For performing Inference-Time Architecture Search (ITAS), we included the related information in Sections 3.3 and A.6. We also included our code in the submission supplementary materials.

A APPENDIX

A.1 ARCHON LLM COMPONENTS

Inference-Time Technique	Definition	Input	Output	Inference Cost	Domains
Generator	Generates a candidate response from an instruction prompt	Instruction Prompt	Candidate Response(s)	1 call per cand.	All Domains
Fuser	Merges multiple candidate responses into a single response	Instruction Prompt + Candidate Response(s)	Fused Candidate Response(s)	1 call per cand.	All Domains
Critic	Generates strengths/weaknesses for each candidate response	Instruction Prompt + Candidate Response(s)	Candidate Response(s) Strengths/Weaknesses	1 call	All Domains
Ranker	Returns top-K candidate responses	Instruction Prompt + Candidate Response(s)	Ranked Candidate Response(s)	1 call	All Domains
Verifier	Returns the candidate responses with verified reasoning	Instruction Prompt + Candidate Response(s)	Verified Candidate Response(s)	2 calls per cand.	Reasoning Tasks
Unit Test Generator	Generates unit tests to evaluate the candidate responses	Instruction Prompt	Instruction Prompt + Unit Tests	1 call	Reasoning Tasks
Unit Test Evaluator	Uses generated unit tests to evaluate candidate response	Instruction Prompt + Unit Tests + Candidate Response(s)	Scored Candidate Response(s)	1 call per cand.	Reasoning Tasks

Table 2: **Overview of ARCHON's Inference-time Techniques**: Definitions, Inputs, Outputs, Costs, and Application Domains.

Module	Initial Layer Placement	Placement after Initial Layer	>1 Module in Layer	Increase Candidate Responses	Decrease Candidate Responses
Generator	Yes	No	Yes	Yes	No
Fuser	No	Yes	Yes	Yes	Yes
Ranker	No	Yes	No	No	Yes
Critic	No	Yes	No	No	No
Verifier	No	Yes	No	No	Yes
Unit Test Generator	No	Yes	No	No	No
Unit Test Evaluator	No	Yes	No	No	No

Table 3: **Rules of ARCHON Construction**: Allowed combinations of each LLM component from Section 3.1.

```
<instruction here>.
```

Table 4: **Generator Prompt**

You have been provided with a set of responses with their individual critiques of strengths/weaknesses from various open-source models to the latest user query. Your task is to synthesize these responses into a single, high-quality response. It is crucial to critically evaluate the information provided in these responses and their provided critiques of strengths/weaknesses, recognizing that some of it may be biased or incorrect. Your response should not simply replicate the given answers but should offer a refined, accurate, and comprehensive reply to the instruction. Ensure your response is well-structured, coherent, and adheres to the highest standards of accuracy and reliability.
Responses from models:
1. <response #1>
Critique: <critique #1>
2. <response #2>
Critique: <critique #2>
...
N. <response #N>
Critique: <critique #N>
<instruction here>

(a) With Critiques

You have been provided with a set of responses from various open-source models to the latest user query. Your task is to synthesize these responses into a single, high-quality response. It is crucial to critically evaluate the information provided in these responses, recognizing that some of it may be biased or incorrect. Your response should not simply replicate the given answers but should offer a refined, accurate, and comprehensive reply to the instruction. Ensure your response is well-structured, coherent, and adheres to the highest standards of accuracy and reliability.
1. <response #1>
2. <response #2>
...
N. <response #N>
<instruction here>

(b) Without Critiques

Table 5: **Fuser Prompt: Without and With Critiques**

I will provide you with N responses, each indicated by a numerical identifier []. Rank the responses based on their relevance to the instruction: <instruction here>.
[1] <response #1>
[2] <response #2>
...
[N] <response #N>
Instruction: <instruction here>.
Rank the N responses above based on their relevance to the instruction. All the responses should be included and listed using identifiers, in descending order of relevance to the instruction. The output format should be [] > [], e.g., [4] > [2]. Only respond with the ranking results, do not say any word or explain.

Table 6: **Decoder-Based Ranking Prompt**

You are a helpful assistant. I will provide you with N responses, each indicated by a numerical identifier (e.g., [1], [2], etc.). Rank the responses based on their relevance to the instruction: <instruction here>.
[1] <response #1>
[2] <response #2>
...
[N] <response #N>
Instruction: <instruction here>.
Evaluate the N responses above based on their relevance to the instruction. All the responses should be included and listed using identifiers. For each response, start the critique with the numerical identifier (e.g., [1]) followed by the strengths and weaknesses. You must include both strengths and weaknesses, even if there are more of one than the other. At the end of each response's analysis, include two new lines to separate the critiques. Do not include any preface or text after the critiques. Do not include any references to previous critiques within a critique. Start with the analysis for the first response and end with the analysis for the last response. All of the N responses should be included and evaluated using identifiers. Structure each response's analysis as follows:
Strengths:
- <strength #1>
- <strength #2>
- <strength #n>
Weaknesses:
- <weakness #1>
- <weakness #2>
- <weakness #n>

Table 7: **Critic Prompt**

I will provide you with a response indicated by the identifier 'Response'. Provide reasoning for why the response accurately and completely addresses the instruction: `<instruction here>`. Response: `<response>` Instruction: `<instruction here>`. Provide the reasoning for the response above based on its relevance, completeness, and accuracy when compared to the instruction. Do not include any preface or text after the reasoning.

Table 8: **Verifier Prompt**

Instruction Prompt: Given the following query, generate a set of `N` unit tests that would evaluate the correctness of responses to this query. - The unit tests should cover various aspects of the query and ensure comprehensive evaluation. - Each unit test should be clearly stated and should include the expected outcome. - The unit tests should be in the form of assertions that can be used to validate the correctness of responses to the query. - The unit test should be formatted like 'The answer mentions...', 'The answer states...', 'The answer uses...', etc. followed by the expected outcome. - Solely provide the unit tests for the question below. Do not provide any text before or after the list. Only output the unit tests as a list of strings (e.g., ['unit test #1', 'unit test #2', 'unit test #3']). Query: `<instruction here>`

(a) With Unit Test Cap

Instruction Prompt: Given the following query, generate a set of unit tests that would evaluate the correctness of responses to this query. - The unit tests should cover various aspects of the query and ensure comprehensive evaluation. - Each unit test should be clearly stated and should include the expected outcome. - The unit tests should be in the form of assertions that can be used to validate the correctness of responses to the query. - The unit test should be formatted like 'The answer mentions...', 'The answer states...', 'The answer uses...', etc. followed by the expected outcome. - Solely provide the unit tests for the question below. Do not provide any text before or after the list. Only output the unit tests as a list of strings (e.g., ['unit test #1', 'unit test #2', 'unit test #3']). Query: `<instruction here>`

(b) Without Unit Test Cap

Table 9: **Unit Test Generator Prompt: With and Without Unit Test Cap**

Instruction Prompt: Compose an engaging travel blog post about a recent trip to Hawaii, highlighting cultural experiences and must-see attractions. 1. Unit Test #1: The blog post mentions at least two cultural experiences specific to Hawaii. 2. Unit Test #2: The blog post highlights at least three must-see attractions in Hawaii. 3. Unit Test #3: The tone of the blog post is engaging and uses descriptive language that would appeal to readers interested in travel. 4. Unit Test #4: The blog post includes factual information about Hawaii's culture, such as local customs, festivals, or historical facts. 5. Unit Test #5: The blog post contains a clear narrative structure, including an introduction, main body, and a conclusion.

(a) Instruction-Following Query

Instruction Prompt: Alice and Bob have two dice. They roll the dice together, note the sum of the two values shown, and repeat. For Alice to win, two consecutive turns (meaning, two consecutive sums) need to result in 7. For Bob to win, he needs to see an eight followed by a seven. Who do we expect to win this game? 1. Unit Test #1: The response correctly identifies the winning condition for Alice (two consecutive sums of 7). 2. Unit Test #2: The response correctly identifies the winning condition for Bob (a sum of 8 followed by a sum of 7). 3. Unit Test #3: The response explains the probability of achieving two consecutive 7s when rolling two dice. 4. Unit Test #4: The response explains the probability of achieving an 8 followed by a 7 when rolling two dice. 5. Unit Test #5: The response provides a conclusion on who is more likely to win based on the probability analysis.

(b) Reasoning Query

Table 10: **Unit Test Examples**

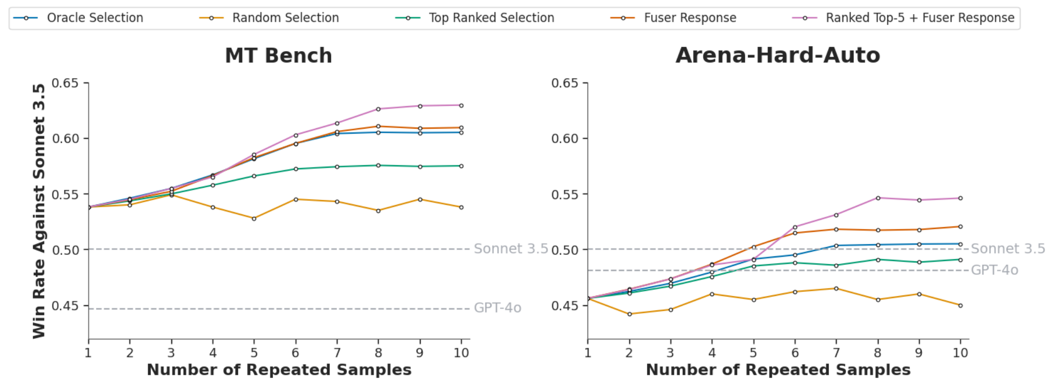

Figure 6: **Performance Gains from Applying Inference Time Techniques on a Single Model**: We repeatedly sample more responses for each individual query. For each sample count, we choose the best response in 5 different ways: **(1)** using an oracle (to get the upper bound for performance of best sample), **(2)** randomly, **(3)** using a ranker model, **(4)** by fusion, in which a model synthesizes a response based on all the samples, and **(5)** by ranking the top-5 best answers and then fusing them. For both MT Bench and Arena-Hard-Auto, we find that fusion is an effective technique. In particular, ranking the candidates first, and then selecting the top-5 and fusing them scores the highest. The best open-source model for these tasks across all the 70B+ models we are considering is WizardLM-2-8x22B (Xu et al., 2024) (see Table 18 for details). For both ranking and fusion, we use Qwen2 72B Instruct (Qwen, 2024).

Given the following query, candidate response, and unit tests, evaluate whether or not the response passes each unit test.
- In your evaluation, you should consider how the response aligns with the unit tests, retrieved documents, and query.
- Provide reasoning before you return your evaluation.
- At the end of your evaluation, you must finish with a list of verdicts corresponding to each unit test.
- You must include a verdict with one of these formatted options: '[Passed]' or '[Failed]'.
- Here is an example of the output format:
Unit Test #1: [Passed]
Unit Test #2: [Failed]
Unit Test #3: [Passed]
- Each verdict should be on a new line and correspond to the unit test in the same position.
- Here is the query, response, and unit tests for your evaluation:

Query: `<instruction here>`.

Candidate Response: `<response>`

Unit Tests:
```
Unit Test #1: <Unit Test #1>
Unit Test #2: <Unit Test #2>
...
Unit Test #N: <Unit Test #N>
```

Table 11: **Unit Test Evaluator Prompt**

A.2 Utilities and Interactions of LLM Components

In this subsection, we present our analysis of the effectiveness of each LLM component (i.e. the *Utility*) and the relationships between each component (i.e. the *Component Interactions*) by evaluating on *instruction-following tasks* (MT Bench, AlpacaEval 2.0, Arena-Hard-Auto), *reasoning tasks* (MixEval, MixEval-Hard, MATH) and *coding tasks* (CodeContests) (Section 4.1). For our Archon models, we utilize a host of 70B+ open-source models (Section 4.1; Table 17).

A.2.1 Generator

Utility: For our Generator module, we find additional model sampling to significantly boost performance (Figure 6), particularly for coding tasks (Table 1). In settings with a limited inference call budget, additional

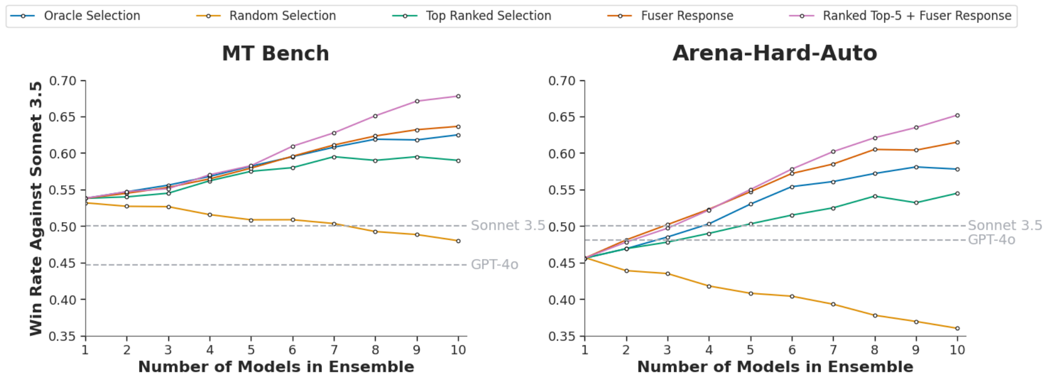

Figure 7: **Performance Gains from Applying Inference-Time Techniques on an Ensemble of Models**: We incrementally add more models to the ensemble, which consists of open-source 70B+ models. The models are added to the pool based on their performance for each task, from best to worse (see Table 18 for details). For each ensemble size, we choose the best response in 5 different modes: **(1)** using an oracle (to get the upper bound for performance of best individual response in the ensemble), **(2)** randomly, **(3)** using a ranker model, **(4)** by fusion, in which one model synthesizes a response based on all the responses of the ensemble models, and **(5)** ranking the top-5 best responses and then fusing them. For MT Bench and Arena-Hard-Auto, we find consistent performance improvements as we add more models to the ensemble. We find that fusion is beneficial across various ensemble sizes and in particular a fused candidate based on the top-5 ranked responses scores highest. The ensemble approach scores higher than applying the same techniques on repeated samples from a single best-performing model (see Figure 6). For both ranking and fusion, we use Qwen2 72B Instruct (Qwen, 2024).

model samples lead to the largest marginal benefit. We see a similar pattern for model ensembling, where sampling from additional models leads to continual performance increases (assuming the models are ordered from best to worst for the given task) (Figure 7).

A.2.2 FUSER

Utility: For every benchmark explored, we found that the Fuser module substantially improved performance (Figure 6; Figure 7; Figure 3). For the single-generation 10-model ensemble of 70B+ models, the Fuser module improved downstream accuracy by 5.2 points, on average, compared to the single-generation best model (Figure 7). When combined with the Ranker module for ranking the top-5 candidate responses, the Fuser improved downstream accuracy by 7.3 points and 3.6 points, on average, compared to the single-sample best model and the oracle best candidate response, respectively (Figure 7). Overall, we found that Fuser efficacy increased as more candidate responses were provided, demonstrating that additional candidate generations can continue to bolster inference-time architecture performance when combined with a Fuser.

In previous work like Mixture-of-Agents (MoA) (Wang et al., 2024), multiple layers of Fusers was found to boost performance on some instruction-following tasks (i.e. MT Bench and Alpaca Eval 2.0). Across all the benchmarks explored, we observed similar benefits in the ARCHON framework when adding multiple layers of Fusers (Figure 3). However, based on our results in Figure 8, the number of Fuser layers needed to improve performance varied by task, with some tasks receiving limited benefits from added layers (1-2 point increase in accuracy for MixEval) while others experienced significant benefits with 3-4 fusion layers and more (2 to 5 point increase in win rate for MT Bench and Alpaca Eval 2.0). We attribute this distinction to the difference in task requirements, with chat and instruction following tasks benefiting more from multiple iterations of revisions through the multiple Fuser layers, leading to greater diversity in the final generation (Table 19).

Component Interactions: To better understand how the Fuser module works with the other LLM components, we took the single-sample 10-model ensemble of Generators with a Fuser and tried adding each of these components individually: a Critic, a Ranker, a Verifier, and a Unit Test Generator/Evaluator. Across all of the benchmarks, the added candidate response analyses from the Critic improved the Fuser's ability to effectively merge the different candidate responses, increasing performance by an average of 3.1 percentage points (Figure 3). With the added Ranker, the ARCHON architecture improved the combined Ensemble + Critic + Fuser performance across all the benchmarks by 4.8 percentage points, on average (Figure 3). The Ranker proved most effective for style-oriented tasks (e.g. MT Bench and AlpacaEval 2.0) since the examples mostly focus on improving the instruction-guidance towards the provided prompt. With the added Verifier module (Figure 3), the performance of the Ensemble + Critic + Fuser configuration improved marginally for the instruction-following tasks (1.2 percentage points, on average, for MT Bench, AlpacaEval 2.0, and Arena-Hard-Auto). However, this configuration improved performance more on reasoning tasks (3.2 percentage points for MixEval and MixEval-Hard, on average), assisting generation by filtering out irrelevant or flawed answers before the final fusion step (Figure 3). The added Unit Test Generator and Evaluator was less effective for the instruction-following and reasoning tasks, only providing a 1.5 percentage points increase, on average, when added to the Ensemble + Critic + Fuser configuration (Table 12). However, for coding tasks, we found unit test generation and evaluation significantly improved performance, leading to a 10.7 percentage point increase (56% performance increase comparatively) as we scale model sampling (Table 1).

A.2.3 Critic

Utility: The Critic module proved effective for every task we explored in Figure 3 and Table 12. With our 10-model 70B+ Generator ensemble and Fuser configuration of Archon, the added Critic improved performance on average by 3.1 percentage points across the benchmarks explored.

Component Interactions: While useful for most Archon architectures, the added strengths and weaknesses from the Critic module are particularly useful when combined with the Fuser module, helping guide generation fusion for a single layer and even useful when placed between multiple fusion layers (on average 3.2 percentage point boost across benchmarks in Figure 3). The Critic module was also effective with the Ranker module, providing additional information for comparing candidate responses (Figure 6) and leading to a 5.9 percentage point increase, on average (Table 12).

A.2.4 Ranker

Utility: From our results in Table 12, Figure 6, and Figure 7, we found the Ranker to be most effective for instruction-following tasks, where pair-wise comparisons of answers focus on style and adherence to the prompt. To examine the candidate selection improvement provided by candidate ranking, we compare three approaches to the Ranker: **(1)** random selection of candidate generation, **(2)** oracle selection of candidate generation, and **(3)** the top-ranked candidate selected by our Ranker. For MT Bench and Arena-Hard-Auto, we find that the ranker improves generation output quality by 3.8% compared to random candidate selection and performs within 2.7% of oracle selection (Figure 6).

Component Interactions: Based on our benchmark results in Table 12, the Ranker pairs well with the Critic module; the provided strengths and weaknesses helps guide ranking, particularly for instruction-following tasks, improving performance by 5.9 percentage points, on average. Furthermore, the Ranker was also effective when paired with the Fuser; the filtered list of candidate responses helped improve the final condensed response produced by the Fuser by 3.8 percentage points, on average (Figure 7). When paired with the Verifier and Unit Test Generator, the Ranker had neutral effects; performances changed marginally, either positively or negatively by 1-2 percentage points (Table 12).

Overall, our findings demonstrate the value of added Rankers for instruction-following and reasoning tasks when paired with Fusers. We find that when Rankers are used alone with an ensemble of Generators, their performance lags behind the 10-sample best single model configuration by 3.0 percentage points, on average (Table 12). Additionally, our findings show the importance of building better rankers for more complex reasoning tasks, such as math and coding, which is a challenge also raised by Brown et al. (2024).

A.2.5 Verifier

Utility: The Verifier was most effective for the reasoning benchmarks explored in Table 12. When just using a 70B+ Generator ensemble with Verifier module after generation, the Archon configuration lagged behind the Archon ensemble and fuser configuration by 1.5 percentage points, on average, across all benchmarks explored. This suggests that the Verifier is most effective when combined with other inference-time techniques.

Component Interactions: As noted in Section A.2.2, the Verifier augmented the performance of the Critic and Fuser on reasoning tasks (e.g. Arena-Hard-Auto, MixEval, MixEval-Hard), boosting performance by 3.7 percentage points, on average, when combined together with these modules. Overall, the Verifier is most powerful when augmenting additional components for tasks requiring verification of intermediate steps and the final response (Table 12). Therefore, the Verifier was less helpful for instruction-following tasks (e.g. MT Bench and AlpacaEval) but more effective for reasoning tasks (e.g. Arena-Hard-Auto and MixEval).

	Model / LLM System	# of Infer. Calls	MT Bench W.R.	AlpacaEval 2.0 L.C. W.R.	AlpacaEval 2.0 Raw W.R.	Arena Hard Auto W.R.	MixEval Hard Acc.	MixEval Acc.	MATH Acc.	Code Contests Acc.
Control	Best Open-Source 70B+ Model, Sampled Once	1	55.0% ±0.4	44.7% ±0.5	37.1% ±0.6	45.6% ±0.5	58.7% ±0.2	86.5% ±0.3	73.5% ±0.6	**27.1% ±0.4**
	Ensemble + *Fuser*	9	58.4% ±0.6	57.5% ±0.4	51.3% ±0.5	54.3% ±0.7	60.5% ±0.3	87.3% ±0.2	75.5% ±0.3	22.0% ±0.7
	Ensemble + *Critic* + *Fuser*	10	60.9% ±0.3	58.7% ±0.6	65.8% ±0.3	58.8% ±0.4	62.4% ±0.4	87.4% ±0.3	77.0% ±0.5	24.5% ±0.5
Ablations	Ensemble + *Ranker*	9	52.5% ±0.7	54.7% ±0.5	47.6% ±0.4	50.5% ±0.6	58.2% ±0.2	86.8% ±0.4	71.5% ±0.4	23.5% ±0.6
	Ensemble + *Verifier*	24	53.2% ±0.5	56.2% ±0.3	50.2% ±0.7	52.4% ±0.3	56.5% ±0.3	85.6% ±0.2	76.0% ±0.7	24.9% ±0.3
	Ensemble + *Unit Test Gen./Eval.*	18	51.5% ±0.4	54.4% ±0.6	49.4% ±0.5	46.1% ±0.8	55.2% ±0.4	86.0% ±0.3	75.0% ±0.5	25.1% ±0.4
	Ensemble + *Ranker* + *Fuser*	10	62.5% ±0.8	60.3% ±0.4	63.6% ±0.6	57.2% ±0.5	60.1% ±0.2	87.6% ±0.3	76.0% ±0.6	23.6% ±0.5
	Ensemble + *Verifier* + *Fuser*	25	60.5% ±0.3	59.4% ±0.7	58.7% ±0.3	59.2% ±0.4	65.1% ±0.3	87.5% ±0.2	78.0% ±0.4	24.5% ±0.7
	Ensemble + *Unit Test Gen./Eval.* + *Fuser*	17	61.4% ±0.6	58.5% ±0.5	55.1% ±0.4	56.4% ±0.7	62.8% ±0.4	86.9% ±0.3	77.0% ±0.8	26.3% ±0.6
	Ensemble + *Critic* + *Verifier* + *Fuser*	25	61.3% ±0.5	60.0% ±0.3	61.0% ±0.7	59.5% ±0.3	65.5% ±0.2	87.8% ±0.4	78.0% ±0.3	24.8% ±0.4
	Ensemble + *Critic* + *Ranker* + *Fuser*	11	**64.7% ±0.4**	**62.6% ±0.6**	**72.4% ±0.5**	**60.9% ±0.6**	**67.0% ±0.3**	**88.3% ±0.2**	**79.5% ±0.5**	24.1% ±0.5

Table 12: **Impact of Different Compositions of ARCHON's Inference-Time Techniques**: We see increased task performances from adding new LLM components to ARCHON. For CodeContests, we find that there is a single model (Llama 3.1 405B Instruct) that performs considerably better than the rest of the LLMs studied, making it more effective leverage additional model sampling (Table 1). For our ensemble, we use the best 8 open-source 70B+ models for the task (Table 18). For our fuser, critic, ranker, and verifier components, we use the best fuser model found for the task (Table 18). For each evaluation benchmark, we explain its configuration in Table 29 and Section 4.1. The standard error numbers were calculated from 10 independent evaluation runs.

	Model / LLM System	# of Infer. Calls	MT Bench W.R.	AlpacaEval 2.0 L.C. W.R.	Arena Hard Auto W.R.	MixEval Hard Acc.	MixEval Acc.	MATH Acc.	Code Contests Acc.
Control	Single Generation	1	44.2% ±0.6	57.8% ±0.5	48.1% ±0.7	63.4% ±0.3	87.5% ±0.2	73.2% ±0.4	17.9% ±0.3
	Ensemble + *Fuser*	11	53.7% ±0.3	59.5% ±0.6	49.7% ±0.5	65.5% ±0.2	82.0% ±0.3	70.7% ±0.6	16.0% ±0.4
	Ensemble + *Critic* + *Fuser*	12	56.1% ±0.7	59.7% ±0.4	53.9% ±0.6	67.4% ±0.4	82.0% ±0.2	71.8% ±0.5	18.9% ±0.6
Ablations	Ensemble + *Ranker*	11	47.6% ±0.4	49.7% ±0.5	45.5% ±0.4	63.3% ±0.3	81.6% ±0.4	66.5% ±0.7	17.9% ±0.5
	Ensemble + *Verifier*	11	48.4% ±0.5	51.2% ±0.7	47.7% ±0.8	61.4% ±0.2	80.5% ±0.3	71.0% ±0.3	23.0% ±0.4
	Ensemble + *Unit Test Gen./Eval.*	21	46.8% ±0.8	49.3% ±0.3	41.2% ±0.5	60.2% ±0.4	80.7% ±0.2	69.9% ±0.8	24.0% ±0.7
	Ensemble + *Ranker* + *Fuser*	12	58.0% ±0.2	60.1% ±0.6	52.2% ±0.3	65.0% ±0.3	82.0% ±0.4	71.0% ±0.4	18.0% ±0.3
	Ensemble + *Verifier* + *Fuser*	12	55.8% ±0.6	54.2% ±0.4	60.3% ±0.7	67.0% ±0.2	82.5% ±0.3	73.1% ±0.6	22.4% ±0.5
	Ensemble + *Unit Test Gen./Eval.* + *Fuser*	22	56.5% ±0.3	61.4% ±0.5	51.6% ±0.4	67.7% ±0.4	81.7% ±0.2	72.0% ±0.5	**25.4% ±0.6**
	Ensemble + *Critic* + *Verifier* + *Fuser*	13	56.6% ±0.7	62.0% ±0.3	55.0% ±0.6	68.5% ±0.3	82.7% ±0.4	73.5% ±0.3	22.2% ±0.4
	Ensemble + *Critic* + *Ranker* + *Fuser*	13	**60.0% ±0.4**	**62.8% ±0.6**	**56.2% ±0.5**	**69.4% ±0.2**	**88.5% ±0.3**	**75.0% ±0.7**	18.5% ±0.5

Table 13: **ARCHON Component Compositions with GPT-4o**: The ensemble uses generates 10 samples for the given query. The standard error numbers were calculated from 10 independent evaluation runs.

	Model / LLM System	# of Infer. Calls	MT Bench W.R.	AlpacaEval 2.0 L.C. W.R.	Arena Hard Auto W.R.	MixEval Hard Acc.	MixEval Acc.	MATH Acc.	Code Contests Acc.
Control	Single Generation	1	32.1% ±0.7	38.5% ±0.5	30.4% ±0.6	45.2% ±0.3	69.5% ±0.2	61.0% ±0.5	10.5% ±0.6
	Ensemble + *Fuser*	11	44.2% ±0.3	43.0% ±0.6	40.2% ±0.4	46.0% ±0.4	73.0% ±0.3	61.2% ±0.7	6.0% ±0.4
	Ensemble + *Critic* + *Fuser*	12	46.6% ±0.5	44.2% ±0.4	44.4% ±0.7	47.9% ±0.2	73.0% ±0.4	62.3% ±0.3	8.4% ±0.5
Ablations	Ensemble + *Ranker*	11	38.1% ±0.6	40.2% ±0.7	36.0% ±0.5	43.8% ±0.3	72.1% ±0.2	57.0% ±0.6	7.5% ±0.4
	Ensemble + *Verifier*	11	38.9% ±0.4	41.7% ±0.3	38.2% ±0.8	41.9% ±0.4	71.0% ±0.3	61.0% ±0.4	19.0% ±0.7
	Ensemble + *Unit Test Gen./Eval.*	21	37.3% ±0.8	39.8% ±0.6	31.7% ±0.3	40.7% ±0.2	71.2% ±0.4	60.4% ±0.8	22.0% ±0.3
	Ensemble + *Ranker* + *Fuser*	12	48.0% ±0.2	45.6% ±0.5	42.7% ±0.6	45.0% ±0.3	73.0% ±0.2	61.0% ±0.5	8.0% ±0.6
	Ensemble + *Verifier* + *Fuser*	12	46.3% ±0.5	44.7% ±0.4	45.0% ±0.4	50.5% ±0.4	73.0% ±0.3	63.6% ±0.3	18.6% ±0.5
	Ensemble + *Unit Test Gen./Eval.* + *Fuser*	22	47.0% ±0.3	43.9% ±0.7	42.1% ±0.7	48.2% ±0.2	72.2% ±0.4	62.0% ±0.6	**23.5% ±0.4**
	Ensemble + *Critic* + *Verifier* + *Fuser*	13	47.1% ±0.7	46.0% ±0.3	45.0% ±0.5	52.4% ±0.3	73.2% ±0.5	63.5% ±0.4	18.4% ±0.7
	Ensemble + *Critic* + *Ranker* + *Fuser*	13	**50.5% ±0.4**	**48.3% ±0.6**	**46.7% ±0.3**	**55.1% ±0.4**	**73.7% ±0.3**	**65.0% ±0.5**	8.1% ±0.5

Table 14: **ARCHON Component Compositions with GPT-4o-mini**: The ensemble uses generates 10 samples for the given query. The standard error numbers were calculated from 10 independent evaluation runs.

	Model / LLM System	# of Infer. Calls	MT Bench W.R.	AlpacaEval 2.0 L.C. W.R.	Arena Hard Auto W.R.	MixEval Hard Acc.	MixEval Acc.	MATH Acc.	Code Contests Acc.
Control	Single Generation	1	N/A	52.7% ±0.4	81.4% ±0.6	68.7% ±0.3	89.1% ±0.2	73.1% ±0.5	12.5% ±0.3
	Ensemble + *Fuser*	11	N/A	53.0% ±0.6	83.2% ±0.4	69.5% ±0.2	89.0% ±0.3	71.2% ±0.6	17.0% ±0.4
	Ensemble + *Critic* + *Fuser*	12	N/A	54.2% ±0.3	85.4% ±0.7	70.9% ±0.4	89.5% ±0.2	72.3% ±0.4	19.4% ±0.6
Ablations	Ensemble + *Ranker*	11	N/A	50.2% ±0.5	76.0% ±0.5	63.8% ±0.3	82.1% ±0.4	67.0% ±0.7	18.5% ±0.5
	Ensemble + *Verifier*	11	N/A	51.7% ±0.7	78.2% ±0.3	60.9% ±0.2	81.0% ±0.3	71.0% ±0.3	21.0% ±0.4
	Ensemble + *Unit Test Gen./Eval.*	21	N/A	49.8% ±0.4	71.7% ±0.8	58.7% ±0.4	81.2% ±0.2	70.4% ±0.8	22.0% ±0.7
	Ensemble + *Ranker* + *Fuser*	12	N/A	55.6% ±0.5	82.7% ±0.4	65.0% ±0.3	89.0% ±0.4	71.0% ±0.4	19.0% ±0.3
	Ensemble + *Verifier* + *Fuser*	12	N/A	54.7% ±0.3	85.0% ±0.6	70.5% ±0.2	89.3% ±0.3	73.6% ±0.6	21.6% ±0.5
	Ensemble + *Unit Test Gen./Eval.* + *Fuser*	22	N/A	53.9% ±0.6	82.1% ±0.5	68.2% ±0.4	89.2% ±0.2	72.0% ±0.5	**23.5% ±0.6**
	Ensemble + *Critic* + *Verifier* + *Fuser*	13	N/A	56.0% ±0.4	85.0% ±0.3	71.0% ±0.3	89.4% ±0.4	73.5% ±0.3	21.4% ±0.4
	Ensemble + *Critic* + *Ranker* + *Fuser*	13	N/A	**58.3% ±0.5**	**86.7% ±0.7**	**73.0% ±0.2**	**89.7% ±0.3**	**75.0% ±0.7**	19.1% ±0.5

Table 15: **ARCHON Component Compositions with Claude 3.5 Sonnet**: The ensemble uses generates 10 samples for the given query. The standard error numbers were calculated from 10 independent evaluation runs.

	Model / LLM System	# of Infer. Calls	MT Bench W.R.	AlpacaEval 2.0 L.C. W.R.	Arena Hard Auto W.R.	MixEval Hard Acc.	MixEval Acc.	MATH Acc.	Code Contests Acc.
Control	Single Generation	1	35.0% ±0.5	42.0% ±0.6	36.8% ±0.7	64.6% ±0.2	73.2% ±0.3	64.8% ±0.4	10.0% ±0.5
	Ensemble + *Fuser*	11	48.2% ±0.3	47.0% ±0.4	44.2% ±0.5	66.5% ±0.3	77.0% ±0.2	65.2% ±0.7	10.8% ±0.3
	Ensemble + *Critic* + *Fuser*	12	50.6% ±0.7	48.2% ±0.5	48.4% ±0.3	68.1% ±0.4	77.0% ±0.4	66.3% ±0.5	11.5% ±0.6
Ablations	Ensemble + *Ranker*	11	42.1% ±0.4	44.2% ±0.7	40.0% ±0.6	58.8% ±0.3	76.1% ±0.2	61.0% ±0.6	11.9% ±0.4
	Ensemble + *Verifier*	11	42.9% ±0.6	45.7% ±0.3	42.2% ±0.8	57.9% ±0.2	75.0% ±0.3	65.0% ±0.4	12.0% ±0.7
	Ensemble + *Unit Test Gen./Eval.*	21	41.3% ±0.8	43.8% ±0.6	35.7% ±0.4	55.7% ±0.4	75.2% ±0.2	64.4% ±0.8	13.0% ±0.3
	Ensemble + *Ranker* + *Fuser*	12	52.0% ±0.2	49.6% ±0.5	46.7% ±0.7	60.0% ±0.3	77.0% ±0.4	65.0% ±0.5	12.0% ±0.6
	Ensemble + *Verifier* + *Fuser*	12	50.3% ±0.5	48.7% ±0.4	48.7% ±0.5	67.5% ±0.2	77.0% ±0.3	67.6% ±0.3	10.5% ±0.5
	Ensemble + *Unit Test Gen./Eval.* + *Fuser*	22	51.0% ±0.3	47.9% ±0.7	46.1% ±0.6	64.2% ±0.4	76.2% ±0.2	66.0% ±0.6	**14.3% ±0.4**
	Ensemble + *Critic* + *Verifier* + *Fuser*	13	51.1% ±0.7	50.0% ±0.3	49.0% ±0.4	68.0% ±0.3	77.2% ±0.4	67.5% ±0.3	10.0% ±0.7
	Ensemble + *Critic* + *Ranker* + *Fuser*	13	**54.5% ±0.4**	**52.3% ±0.6**	**50.7% ±0.3**	**70.4% ±0.2**	**77.7% ±0.3**	**69.0% ±0.5**	11.5% ±0.5

Table 16: **ARCHON Component Compositions with Claude-3-Haiku**: The ensemble uses generates 10 samples for the given query. The standard error numbers were calculated from 10 independent evaluation runs.

A.2.6 UNIT TEST GENERATOR AND EVALUATOR

Utility: The Unit Test Generator and Evaluator were most effective on reasoning and coding tasks, improving performance on benchmarks that required more verification steps, such as Arena-Hard-Auto, MixEval, MixEval-Hard, MATH, and CodeContests (Table 12). For the reasoning tasks, we found the unit test generator and evaluator to be most effective when combined with other components. When the 70B+ ensemble of Generators was only combined with unit tests, it was less effective for reasoning tasks like Arena-Hard-Auto and MixEval, lagging behind the ensemble and fuser configuration by 3.1 percentage points. This inspired us to look into other inference-time techniques combinations for unit test generation, such as increased sampling and fusion. When we increased generation sampling and added unit test generation/evaluation for CodeContests, we see a 56% boost in Pass@1 performance (Table 1), increasing from 17.9 to 29.3 Pass@1.

Component Interactions: When combined with the Fuser module, the Unit Test Generator and Evaluator improved performance by 2.1 percentage points across the benchmarks explored (Table 12). The combined ensemble, Unit Test Generator/Evaluator, and Fuser ARCHON configuration was most effective on the reasoning benchmarks, leading to a 2.5 percentage point boost, on average. For coding, the unit test generator and evaluator was most effective when combined with the best performing Generator (using large sample counts) and a final Fuser (subsection 4.2).

Models	MT Bench Gen	MT Bench Fusion	Alpaca Eval 2.0 Gen	Alpaca Eval 2.0 Fusion	Arena Hard Auto Gen	Arena Hard Auto Fusion	MixEval Gen	MixEval Fusion	MixEval Hard Gen	MixEval Hard Fusion	MATH Gen	MATH Fusion	CodeContests Gen	CodeContests Fusion
GPT-4o	44.7%	61.9%	**57.5%**	64.5%	**48.1%**	69.2%	88.0%	**89.4%**	63.6%	65.4%	72.0%	75.5%	17.9%	19.4%
GPT-4-Turbo	42.2%	**63.1%**	55.0%	**65.8%**	**48.1%**	61.9%	88.9%	89.0%	64.1%	64.4%	74.5%	76.5%	9.3%	14.2%
Claude 3 Opus	30.9%	57.2%	40.5%	N/A	27.0%	47.9%	88.3%	88.2%	63.6%	64.0%	72.5%	71.0%	10.0%	12.5%
Claude 3.5 Sonnet	N/A	71.9%	52.37%	63.6%	N/A	**73.2%**	**89.7%**	89.3%	**68.9%**	**69.5%**	72.0%	74.5%	12.1%	15.5%
Qwen 2 72B Instruct	35.0%	59.7%	37.48%	56.0%	14.5%	49.5%	86.5%	87.5%	58.7%	61.1%	**76.0%**	**78.5%**	3.6%	5.2%
DeepSeek LLM 67B Instruct	18.4%	20.0%	17.8%	17.1%	N/A	N/A	79.2%	N/A	42.5%	N/A	45.0%	N/A	5.7%	N/A
Qwen 1.5 72B Chat	24.7%	46.3%	36.6%	55.7%	14.4%	36.4%	84.5%	82.1%	50.3%	52.2%	62.5%	65.5%	15.0%	13.9%
Qwen 1.5 110B Chat	34.4%	50.3%	43.6%	55.9%	21.9%	39.7%	85.3%	86.5%	51.8%	55.6%	67.0%	72.5%	3.6%	7.8%
Wizard 8x22B	**53.8%**	57.2%	44.7%	50.6%	45.6%	51.2%	83%	78.1%	54.3%	50.4%	69.0%	58.5%	7.1%	10.4%
Llama 3.1 8B Instruct	33.1%	45.9%	25.6%	34.9%	11.9%	28.6%	75.0%	57.5%	41.3%	46.5%	59.0%	60.5%	8.6%	7.8%
Llama 3.1 70B Instruct	45.0%	51.9%	35.6%	40.2%	23.8%	37.2%	85.7%	83.5%	61.1%	65.5%	69.0%	71.5%	20.7%	**23.4%**
Llama 3.1 405B Instruct	44.7%	N/A	40.3%	N/A	28.4%	N/A	88.9%	N/A	66.2%	N/A	74.5%	N/A	**27.1%**	N/A

Table 18: **ARCHON Generation and Fusion Performances for Single Models**: For Alpaca Eval 2.0, we use the length-controlled win rate (LC WR). For fusion, we gather one candidate from each of the top-10 generator models.

A.3 ARCHON LLM ANALYSIS

Model	Source Code	Parameter Count	Max Sequence Length
GPT-4o (OpenAI et al., 2024)	Closed-Source	—	128K
GPT-4-Turbo (OpenAI et al., 2024)	Closed-Source	—	128K
Claude-3-Opus (Anthropic, 2024)	Closed-Source	—	200K
Claude-3.5-Sonnet (Anthropic, 2024)	Closed-Source	—	200K
Claude-3-Haiku (Anthropic, 2024)	Closed-Source	—	200K
Llama-3.1-70B-Instruct (Dubey et al., 2024)	Open-Source	70B	8k
Llama-3.1-405B-Instruct (Dubey et al., 2024)	Open-Source	70B	8k
DeepSeek LLM 67B Chat (Guo et al., 2024)	Open-Source	67B	32k
Qwen2 72B Instruct (Qwen, 2024)	Open-Source	72B	32k
Qwen1.5 110B Chat (Bai et al., 2023)	Open-Source	110B	32k
Qwen1.5 72B Chat (Bai et al., 2023)	Open-Source	72B	32k
Mixtral 8x22B v0.1 (Jiang et al., 2024)	Open-Source	176B	32k
WizardLM 8x22B (Xu et al., 2024)	Open-Source	176B	32k
dbrx-instruct (Databricks, 2024)	Open-Source	132B	32k
princeton-nlp/Llama-3-Instruct-8B-SimPO (Meng et al., 2024)	Open-Source	8B	8k
princeton-nlp/Llama-3-Instruct-8B-DPO (Meng et al., 2024)	Open-Source	8B	8k
princeton-nlp/Llama-3-Instruct-8B-RDPO (Meng et al., 2024)	Open-Source	8B	8k
princeton-nlp/Llama-3-Instruct-8B-IPO (Meng et al., 2024)	Open-Source	8B	8k
Llama-3.1-8B-Instruct (Dubey et al., 2024)	Open-Source	8B	8k
Qwen2-7B-Instruct (Qwen, 2024)	Open-Source	7B	32k
Qwen/Qwen1.5-7B-Chat (Bai et al., 2023)	Open-Source	7B	32k
mistralai/Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a)	Open-Source	7B	32k
cognitivecomputations/dolphin-2.2.1-mistral-7b (Hartford, 2024)	Open-Source	7B	32k
microsoft/Phi-3-mini-4k-instruct (Abdin et al., 2024)	Open-Source	4B	4k
HuggingFaceH4/zephyr-7b-beta (Tunstall et al., 2023)	Open-Source	7B	32k
microsoft/Phi-3-small-8k-instruct (Abdin et al., 2024)	Open-Source	7B	8k
snorkelai/Snorkel-Mistral-PairRM-DPO (Tran et al., 2023)	Open-Source	7B	32k
mistralai/Mistral-7B-Instruct-v0.3 (Jiang et al., 2023a)	Open-Source	7B	32k

Table 17: **Models Tested with ARCHON**.

Inference-Time Architecture	Jaccard Similarity (%)						
	MT Bench	AlpacaEval 2.0	Arena-Hard Auto	MixEval	MixEval Hard	MATH	Code Contests
Best Open-Source 70B+ Model, Sampled 8 Times + Fuser	45.3%	52.1%	48.4%	55.2%	58.9%	65.2%	63.7%
Ensemble (8 Top Models), Sampled Once Each + Fuser	31.6%	34.1%	28.9%	38.6%	40.9%	57.1%	53.4%

Table 19: **Jaccard Similarities between Candidates Responses and Fused Response by Benchmark**: For the fuser, we use the best-performing 70B+ model for benchmark.

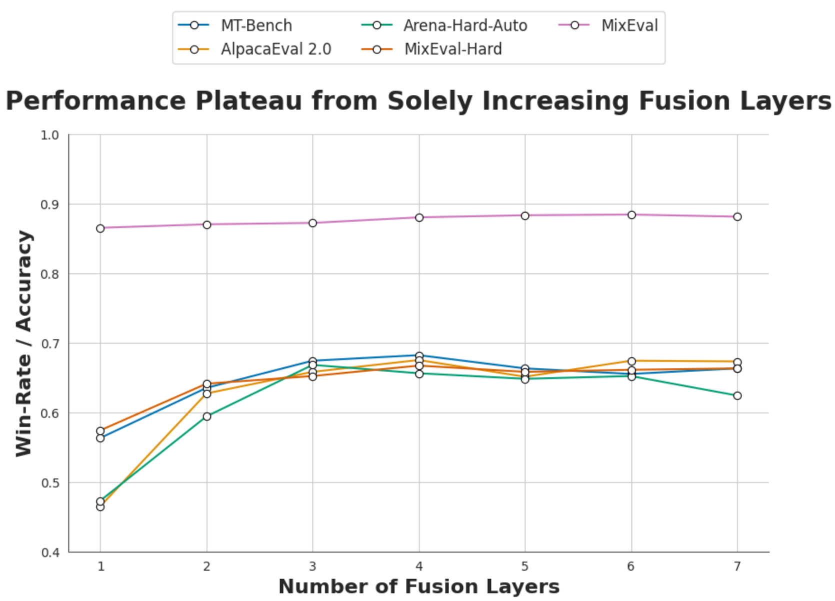

Figure 8: **Fusion Layer Efficacy by Benchmark**: From solely scaling the fusion layers, we see limited benefits across the benchmarks explored but when we add other inference-time techniques, such as Critic and Ranker, we see increased downstream performance as we continue scaling inference-time compute (Figure 3). We use an 8-model ensemble of the top Generator models for each benchmark (Table 18). For our Fuser layers, we use the best Fuser model for the final fuser layer (Table 18). For the intermediate layers, we use the top-8 Fuser models for each benchmark.

A.4 ARCHON ARCHITECTURES

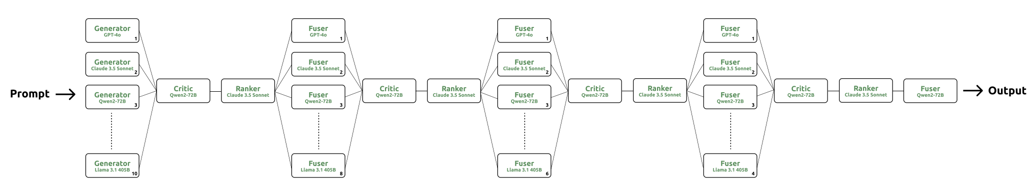

Figure 9: **All-Source Generalizable ARCHON Architecture**: Using ITAS, we found this all-source ARCHON configuration to be effective across the benchmarks explored (except for CodeContests). In the diagram above, we use 10 SOTA all-source LLMs to create multiple successive layers of critic, ranker, and fusers, with each successive fuser layer having less fusers to produce a "funneling" effect as the candidate generations are processed. The layers of critic, ranker, and fuser led to better candidate generations through iterative critique and rewriting. Each of the initial Generator models were sampled once.

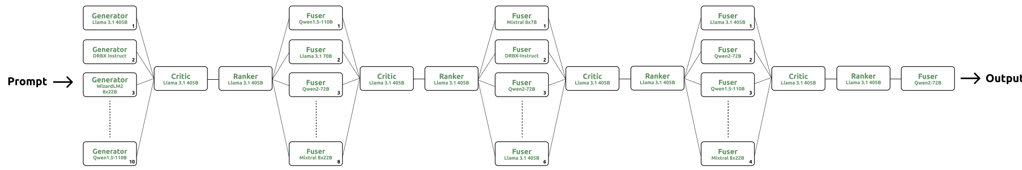

Figure 10: **Open-Source Generalizable ARCHON Architecture**: Using ITAS, we found this open-source ARCHON configuration to be effective across the benchmarks explored (except for CodeContests). In the diagram above, we use 10 SOTA open-source LLMs to create multiple successive layers of critic, ranker, and fusers, with each successive fuser layer having less fusers to produce a "funneling" effect as the candidate generations are processed. The layers of critic, ranker, and fuser led to better candidate generations through iterative critique and rewriting. Each of the initial Generator models were sampled once.

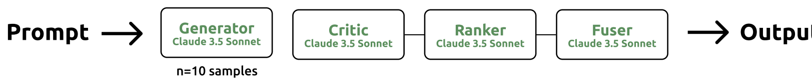

Figure 11: **All-Source ARCHON Architecture for Instruction-Following**: Using ITAS, we found Claude-3.5-Sonnet as a generator, critic, ranker, and fuser to be an effective targeted architecture for instruction-following tasks, such as MT Bench and AlpacaEval 2.0. The ranker picks the top-5 candidate responses to send to the fuser. Each of the initial Generator models were sampled once.

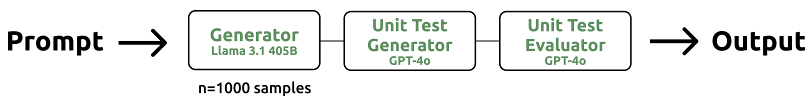

Figure 12: **All-Source ARCHON Architecture for CodeContests**: Using ITAS, we were able to get improved code generation on CodeConetsts by utilizing Llama 3.1 405B for generation and GPT-4o for unit testing (Figure 5). The unit test generator produces 10 unit tests for evaluation. Each of the initial Generator models were sampled once.

A.5 ARCHON BY INFERENCE COMPUTE BUDGET, MODEL SIZE, AND COST

		Datasets				
	Number of Inference Calls	MT Bench	Alpaca Eval 2.0	Arena Hard Auto	MixEval	MixEval Hard
70B+ Models	1	55.0%	44.7%	45.6%	86.5%	61.1%
	10	52.5%	50.6%	45.6%	86.5%	63.9%
	20	65.3%	60.4%	59.4%	89.0%	65.0%
	30	69.2%	64.5%	**69.0%**	**89.5%**	**67.5%**
	40	69.5%	**66.7%**	**69.0%**	**89.5%**	**67.5%**
	50	**71.6%**	**66.7%**	**69.0%**	**89.5%**	**67.5%**
Closed Models	1	45.0%	57.5%	48.1%	88.9%	68.9%
	10	57.1%	63.2%	68.4%	90.0%	70.1%
	20	59.4%	66.5%	75.5%	**90.6%**	70.5%
	30	70.2%	**68.8%**	**77.4%**	**90.6%**	**72.9%**
	40	75.5%	**68.8%**	**77.4%**	**90.6%**	**72.9%**
	50	**80.4%**	**68.8%**	**77.4%**	**90.6%**	**72.9%**

Table 20: **ARCHON with Different Inference Budgets**: For AlpacaEval 2.0, we use the length-controlled win rate (LC WR).

Models / LLM Systems	MT Bench	Alpaca Eval 2.0	Arena Hard Auto	MixEval	MixEval Hard
SOTA Single-Model	**44.7%**	**57.5%**	**48.1%**	68.9%	**89.7%**
Best Model, 1-Sample	15.7%	41.0%	18.3%	76.2%	46.1%
Best Model - 10-Sample + Ranking	16.5%	43.2%	18.9%	78.4%	48.5%
10-Model, 1-Sample Ensemble + Ranking	22.4%	48.2%	25.6%	**81.5%**	52.9%
10-Model, 1-Sample Ensemble + Fusion	14.3%	39.4%	17.5%	73.2%	45.2%
10-Model, 1-Sample Ensemble + Top-5 Ranking + Fusion	15.9%	41.2%	18.0%	75.1%	46.9%
10-Model, 1-Sample Ensemble + Critic + Fusion	10.5%	38.4%	16.5%	71.4%	42.5%

Table 21: **ARCHON with 7B Open-Source Models**: For AlpacaEval 2.0, we use the length-controlled win rate (LC WR). We use open-source 7B models for testing from Table 17.

Models	Cost ($) per Million Input Tokens	Cost ($) per Million Output Tokens
Claude 3.5 Sonnet	$3	$15
Claude 3.0 Opus	$15	$75
GPT-4o	$5	$15
GPT-4-Turbo	$10	$30
TogetherAI - Llama 3.1 405B Instruct	$5	$5
TogetherAI - Llama 3.1 70B Instruct	$0.88	$0.88
TogetherAI - Other Models	$0.90	$0.90

Table 22: **Model API Costs as of August 2024**

Model / LLM System	Cost ($) per Query for Benchmark						
	MT Bench	AlpacaEval 2.0	Arena-Hard Auto	MixEval	MixEval Hard	MATH	Code Contests
Claude 3.5 Sonnet	0.0305	0.0171	0.0212	0.0231	0.0226	0.0325	0.384
GPT-4o	0.0481	0.0236	0.0324	0.0357	0.0361	0.514	0.562
Llama 3.1 405B Instruct	0.0281	0.0174	0.0185	0.0212	0.0205	0.305	0.372
General Purpose ARCHON Architecture	0.364	0.189	0.195	0.284	0.252	0.375	0.461
Task Specific ARCHON Architecture	0.401	0.210	0.221	0.295	0.265	0.425	0.448

Table 23: **ARCHON Costs per Query by Benchmark**

A.6 BAYESIAN OPTIMIZATION

Bayesian Optimization is a sequential design strategy for global optimization of black-box functions that are expensive to evaluate Snoek et al. (2012). It is particularly useful when dealing with functions that have unknown forms and are costly to evaluate, such as hyperparameter tuning in machine learning.

A.6.1 OVERVIEW OF BAYESIAN OPTIMIZATION

The core idea behind Bayesian Optimization is to build a probabilistic model of the objective function and use it to select the most promising points to evaluate next. This process involves two main components:

1. **Surrogate Model**: A probabilistic model (often a Gaussian Process) that approximates the unknown objective function.

2. **Acquisition Function**: A function that guides the search for the optimum by suggesting the next point to evaluate, based on the surrogate model.

A.6.2 STEPS IN BAYESIAN OPTIMIZATION

1. **Initialization**: Begin with a set of initial points $\mathcal{D}=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),...,(\mathbf{x}_n,y_n)\}$, where $\mathbf{x}_i$ is the input, and $y_i=f(\mathbf{x}_i)$ is the objective function value at $\mathbf{x}_i$.
2. **Model Building**: Fit a surrogate model (e.g., Gaussian Process) to the observed data $\mathcal{D}$.
3. **Acquisition**: Use the acquisition function to select the next point $\mathbf{x}_{n+1}$ to evaluate:

$$\mathbf{x}_{n+1}=\arg\max_{\mathbf{x}} a(\mathbf{x}|\mathcal{D})$$

   where $a(\mathbf{x}|\mathcal{D})$ is the acquisition function.
4. **Evaluation**: Evaluate the objective function at $\mathbf{x}_{n+1}$ to get $y_{n+1}=f(\mathbf{x}_{n+1})$.
5. **Update**: Add the new data point $(\mathbf{x}_{n+1},y_{n+1})$ to the dataset $\mathcal{D}$.
6. **Repeat**: Repeat steps 2-5 until convergence or a stopping criterion is met (e.g., budget exhausted, no significant improvement).

A.6.3 GAUSSIAN PROCESS AS A SURROGATE MODEL

A Gaussian Process (GP) is commonly used as a surrogate model in Bayesian Optimization. It is defined by a mean function $\mu(\mathbf{x})$ and a covariance function (kernel) $k(\mathbf{x},\mathbf{x}')$:

$$f(\mathbf{x})\sim\mathcal{GP}(\mu(\mathbf{x}),k(\mathbf{x},\mathbf{x}'))$$

Given a set of observations $\mathcal{D}$, the GP provides a predictive distribution for the objective function at a new point $\mathbf{x}$:

- **Predictive Mean**: The expected value of the function at $\mathbf{x}$:

$$\mu(\mathbf{x}|\mathcal{D})=\mathbf{k}_n(\mathbf{x})^T\mathbf{K}_n^{-1}\mathbf{y}$$

  where $\mathbf{k}_n(\mathbf{x})$ is the covariance vector between $\mathbf{x}$ and the training points, and $\mathbf{K}_n$ is the covariance matrix of the training points.
- **Predictive Variance**: The uncertainty in the function value at $\mathbf{x}$:

$$\sigma^2(\mathbf{x}|\mathcal{D})=k(\mathbf{x},\mathbf{x})-\mathbf{k}_n(\mathbf{x})^T\mathbf{K}_n^{-1}\mathbf{k}_n(\mathbf{x})$$

A.6.4 ACQUISITION FUNCTIONS

Acquisition functions guide the search for the optimum by balancing exploration (trying out areas with high uncertainty) and exploitation (focusing on areas with high predicted values). Common acquisition functions include:

1. **Expected Improvement (EI)**:

$$\text{EI}(\mathbf{x})=\mathbb{E}[\max(0,f(\mathbf{x})-f(\mathbf{x}^+))]$$

   where $f(\mathbf{x}^+)$ is the best observed value so far.

2. **Probability of Improvement (PI)**:

$$\mathrm{PI}(\mathbf{x}) = \mathbb{P}(f(\mathbf{x}) > f(\mathbf{x}^+) + \xi)$$

where ξ is a small positive number.

3. **Upper Confidence Bound (UCB)**:

$$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}|\mathcal{D}) + \kappa\sigma(\mathbf{x}|\mathcal{D})$$

where κ controls the trade-off between exploration and exploitation.

A.6.5 Summary of Bayesian Optimization

Bayesian Optimization iteratively uses a surrogate model to approximate the objective function and an acquisition function to decide where to sample next. By focusing on promising areas of the search space and systematically exploring uncertain regions, it efficiently optimizes complex, expensive-to-evaluate functions.

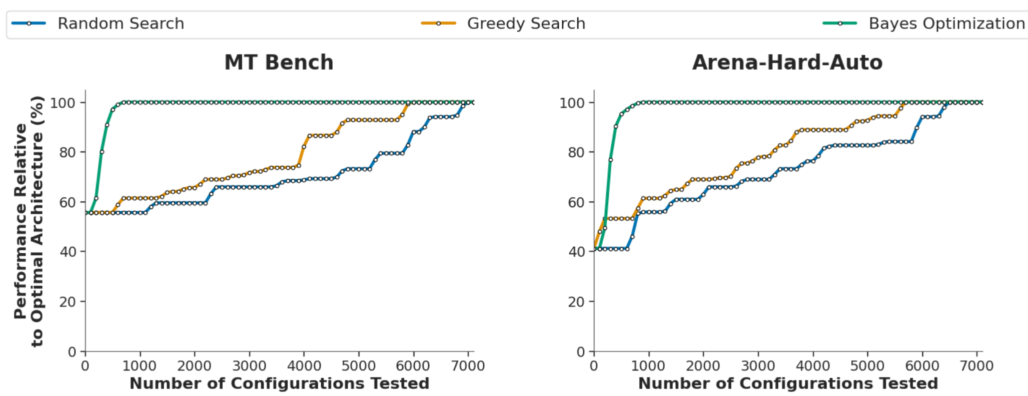

Figure 13: **Impact of Different Optimization Algorithms on Inference-Time Architecture Search (ITAS)**: On the benchmarks MT Bench and Arena-Hard-Auto, we compare four approaches for finding the optimal inference-time architecture: random search, greedy search, and Bayes Optimization. Bayes Optimization finds the optimal architecture in 88.5% less iterations compared to greedy search and 90.4% less iterations compared to random search.

A.7 Bayes Optimization vs. Alternative Approaches

Search Techniques: Within the hyperparameter space, we explored three search algorithms for automating the development of inference-time architectures:

1. **Random Search**: Randomly selects a combination of hyperparameters for our ARCHON architecture.
2. **Greedy Search**: Starting with a base ARCHON configuration, marginally changes each hyperparameter and test if it improves performance or not. If it does, incorporate the change. If not, move on to the next hyperparameter.
3. **Bayesian Optimization**: Efficiently selects the most promising hyperparameter configurations for ARCHON by building a probabilistic surrogate model and leveraging an acquisition function for hyperparameter selection (Snoek et al., 2012; Nardi et al., 2019) (Section A.6).

To get our model ranking for the benchmark, we calculate the model ranking by testing each model individually on a 20% sample of each dataset benchmark in the first stage of the search. To get our fusion model ranking for the benchmark, we use the same approach, testing each model's fusion performance with an ensemble of 10 randomly selected models from the available set. From our experiments, we found that the best generator and fusion models could vary widely dataset to dataset, making it beneficial to perform these rankings for new datasets (Table 18). For search, we use the same 20% sample of each dataset that was used for evaluating generation and fusion, allowing us to guide architecture search with improved evaluation speed while getting meaningful development signal.

Comparing Search Algorithms: In Figure 13, we compare the effectiveness of each search algorithm on our explored benchmarks. While random search guarantees the optimal ARCHON configuration, we found Bayesian optimization to be most effective in terms of tradeoff between finding the optimal configurations and minimizing the number of configurations tested. For 96.0% percent of the search iterations tested in Figure 13, we found that Bayesian optimization had the optimal configuration amongst the four explored search algorithms. We use 230 initial samples for our Bayes Optimization architecture search (Section A.6). Bayesian optimization also found the best architecture configuration in 88.5% less evaluations than greedy search and 90.4% less evaluations than random search.

Bayesian Optimization Analysis: In Table 26, we explore how the number of initial testing points, the number of exploration iterations, and the ARCHON inference call budget impacts the effectiveness of Bayesian optimization. Additional initial testing points continue improving search efficacy up until 230-240 samples, where testing would be better delegated towards configuration search. For lower inference call budgets with ARCHON (e.g. <20 inference calls), Bayesian optimization proved less effective, performing more similarly to greedy search or random search given the limited search space (Table 27). Therefore, Bayesian optimization is more effective for more open-ended ITAS with larger inference call budgets (e.g. >20 inference calls) whereas traditional component engineering might be better for more limited inference call budgets.

A.8 ITAS ALGORITHMS COMPARISONS

# of Init. Points	% of Total Configs	Iter. till Max. Config.	Comb. Iter.
200	2.18%	353	553
210	2.29%	324	534
220	2.40%	301	521
230	2.51%	284	514
240	2.61%	**261**	**501**
250	2.72%	265	515
260	2.83%	256	516
270	2.94%	252	522

Table 24: MT Bench

# of Init. Points	% of Total Configs	Iter. till Max. Config.	Comb. Iter.
200	2.18%	478	678
210	2.29%	431	641
220	2.40%	415	635
230	2.51%	**382**	**612**
240	2.61%	389	629
250	2.72%	385	635
260	2.83%	372	632
270	2.94%	368	638

Table 25: Arena-Hard-Auto

Table 26: **Bayesian Optimization Hyperparameter Comparisons**: On MT Bench and Arena-Hard-Auto, we compare Bayesian optimization configurations for the number of initial sample points. We find that 230 to 240 initial sample points minimizes the combined number of iterations (both initial sampling and exploring) to find the optimal configuration. For the configurations explored, the total number of hyperparameter choices is 9,576.

	Iterations to Convergence				
Inference Budget	10	20	30	40	50
Random Selection	387	1152	2731	4359	5843
Greedy Search	343	984	2153	3045	4895
Bayes Optimization	**254**	**386**	**452**	**515**	**589**

Table 27: **ITAS Algorithms Comparison by Inference Call Budget**: For our comparison, we evaluate on MT Bench.

A.9 ARCHON BENCHMARKS AND RESULTS

			Datasets							
			MT Bench	Alpaca Eval 2.0		Arena Hard Auto	Arena Hard Auto	MixEval Hard	MixEval	MATH*
	Judge Model		GPT-4 0314	GPT-4 Turbo		GPT-4 Turbo	GPT-4 Turbo	N/A	N/A	N/A
	Reference Model		Claude 3.5 Sonnet	GPT-4 Turbo		Claude 3.5 Sonnet	GPT-4 Turbo	N/A	N/A	N/A
	Model / LLM System	**Infer. Calls**	W.R.	L.C. W.R.	Raw W.R.	W.R.	W.R	Acc.	Acc.	Pass @1
	GPT-4o - 2024-05-13	1	44.7%	57.5%	51.3%	48.1%	80.3%	63.6%	88.0%	72.0%
	Claude 3.5 Sonnet	1	N/A	52.4%	40.6%	N/A	80.9%	68.9%	89.7%	72.0%
	Llama 3.1 405B Instruct	1	44.7%	40.3%	37.7%	28.4%	64.1%	66.2%	88.9%	74.0%
	MoA	19	51.6%	65.1%	59.8%	52.2%	84.2%	62.5%	87.3%	72.5%
	MoA Lite	7	45.6%	59.3%	57.0%	40.6%	87.8%	61.1%	87.1%	70.5%
Open Source	General-purpose ARCHON Architecture	35	67.5%	63.0%	68.3%	66.2%	85.1%	65.5%	86.9%	75.5%
Open Source	Task-specific ARCHON Architectures	44	71.6%	66.7%	70.7%	69.0%	89.5%	67.5%	89.6%	**80.5%**
Closed Source	General-purpose ARCHON Architecture	32	73.1%	63.5%	69.1%	70.5%	85.8%	67.7%	88.2%	77.0%
Closed Source	Task-specific ARCHON Architectures	40	77.5%	68.4%	72.1%	74.4%	90.2%	**72.9%**	90.4%	79.0%
All Source	General-purpose ARCHON Architecture	35	76.8%	65.8%	70.2%	72.5%	89.3%	70.1%	88.1%	78.0%
All Source	Task-specific ARCHON Architectures	39	**80.4%**	**67.6%**	**73.3%**	**76.1%**	**92.1%**	**72.9%**	**90.6%**	**80.5%**

Table 28: **ARCHON's Strong Performance on the Complete Evaluation Datasets after ITAS Optimization**: We find that ARCHON's inference-time architectures consistently outperform single-call state-of-the-art LLMs, both open-source and closed-source baselines, when evaluating on the complete benchmarks (Table 29). We explore two configurations: ITAS for building custom ARCHON configurations for each individual benchmark and ITAS for building a single general-purpose ARCHON configuration for all the benchmarks (Section 4.4). We find that a general ARCHON configuration lags behind the custom ones by only 3.2 percentage points, on average, across our all-source settings, which suggests the efficacy of general-purpose inference-time architectures created with our framework. For Arena-Hard-Auto, we also include a configuration with Claude 3.5 Sonnet as a stronger reference model for comparison against ARCHON inference-time architectures and to mitigate bias from GPT judges towards GPT generations. For MT Bench, we use a GPT-4-0314 judge model instead of newer LLM judges to be consistent with previous results on this benchmark. For our task-specific ARCHON architectures, we also provide the average inference calls across the given benchmarks. For our full-list of models explored, please see Table 17. For MATH, we use a randomly sampled subset of size 200 for evaluation (Section 4.1; Table 29). We include our ARCHON architecture results on the held-out 80% subset of each evaluation benchmark in Table 1.

Benchmark	Example Count	Reference Model	Judge Model	Scoring Type	Metric
AlpacaEval 2.0	805	GPT-4-Turbo	GPT-4-Turbo	Pairwise Comparison	L.C. & Raw Win Rates
Arena-Hard-Auto	500	Claude-3.5-Sonnet GPT-4-0314	GPT-4-Turbo	Pairwise Comparison	Win Rate
MT-Bench	80	Claude-3.5-Sonnet	GPT-4-0314	Pairwise Comparison	Adjusted Win Rate
MixEval	2000	N/A	N/A	Ground Truth	Accuracy
MixEval-Hard	500	N/A	N/A	Ground Truth	Accuracy
MATH	200 (sampled from 5000)	N/A	N/A	Ground Truth	Pass@1
CodeContests	140 (non-visual queries)	N/A	N/A	Ground Truth	Pass@1

Table 29: **Benchmark Overview**: Evaluation configurations for AlpacaEval 2.0 (Li et al., 2023), Arena-Hard-Auto (Li et al., 2024b), MT-Bench (Zheng et al., 2023), MixEval (Ni et al., 2024), MixEval Hard, MATH (Hendrycks et al., 2021), and CodeContests (Li et al., 2022)

.

		Arena-Hard-Auto	
	Model / LLM System	Score	C.I.
	Claude 3.5 Sonnet	N/A	N/A
	GPT-4o	48.1%	(-2.3, 1.8)
	Llama 3.1 405B Instruct	28.4%	(-2.7, 2.5)
Open Source	General-purpose ARCHON Architecture	66.2%	(-2.4, 2.2)
	Task-specific ARCHON Architectures	69.0%	(-2.8, 2.5)
Closed Source	General-purpose ARCHON Architecture	70.5%	(-2.5, 2.0)
	Task-specific ARCHON Architectures	74.4%	(-2.3, 1.6)
All Source	General-purpose ARCHON Architecture	72.5%	(-2.5, 1.8)
	Task-specific ARCHON Architectures	**76.1%**	(-1.8, 2.2)

Table 30: **ARCHON Results on Arena-Hard-Auto Results with Claude-3.5-Sonnet as Baseline Model**: The baseline model is Claude-3.5-Sonnet (default baseline model: GPT-4-0314) while the judge model is GPT-4-Turbo.

		MixEval - Sub-Datasets						
Model / LLM System	**Infer. Calls**	GSM8K	TriviaQA	DROP	MATH	BBH	AGIEval	**Average**
GPT-4o - 2024-05-13	1	94.9	89.1	88.2	**98.5**	98.3	71.5	90.3
Claude 3.5 Sonnet	1	98.0	92.0	92.6	96	95.6	78.0	92.0
Llama 3.1 405B Instruct	1	**98.2**	87.9	89.6	91.5	95.8	73.2	89.6
General-purpose ARCHON Architecture	29	98.3	94.8	94.6	98.1	97.3	82.1	94.2
Task-specific ARCHON Architectures	34	**98.2**	**96.7**	**95.6**	**98.5**	**98.8**	**84.2**	**95.7**

Table 31: **MixEval Results by Sub-Dataset**: For the average computed, we do not introduce any weighting for each dataset.

Model / LLM System	Infer. Calls	MixEval - Sub-Datasets						
		GSM8K	TriviaQA	DROP	MATH	BBH	AGIEval	**Average**
GPT-4o - 2024-05-13	1	72.3	70.5	70.2	94.4	80.0	53.5	73.5
Claude 3.5 Sonnet	1	87.3	75.5	79.3	82.5	80.0	74.6	79.9
Llama 3.1 405B Instruct	1	98.7	71.2	70.7	86.9	78.8	62.0	78.1
General-purpose ARCHON Architecture	33	96.7	82.7	83.2	93.4	82.0	76.7	85.8
Task-specific ARCHON Architectures	37	**98.9**	**86.2**	**85.2**	**96.2**	**86.0**	**80.1**	**88.8**

Table 32: **MixEval-Hard Results by Sub-Dataset**: For the average computed, we do not introduce any weighting for each dataset.

	GSM8K	**MMLU Math**	**HumanEval Python**	**MBPP**
Model	Pass@1	Pass@1	Pass@1	Pass@1
GPT-4o	97.1%	84.8%	89.0%	87.5%
Claude 3.5 Sonnet	96.8%	90.9%	90.2%	88.9%
Llama 3.1 405B Instruct	95.9%	85.4%	90.2%	88.6%

Table 33: **Additional Math and Code Benchmarks Explored**

REFERENCES

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

AI@Meta. Llama 3 model card. *ArXiv*, 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

Anthropic. The claude 3 model family: Opus, sonnet, haiku. *ArXiv*, 2024.

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. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022a. URL `https://arxiv.org/abs/2212.08073`.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022b.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL `https://arxiv.org/abs/2407.21787`.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more llm calls all you need? towards scaling laws of compound inference systems, 2024. URL https://arxiv.org/abs/2403.02419.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022. URL https://arxiv.org/abs/2210.11416.

Databricks. Dbrx technical report. 2024.

Jared Quincy Davis, Boris Hanin, Lingjiao Chen, Peter Bailis, Ion Stoica, and Matei Zaharia. Networks of networks: Complexity class principles applied to compound ai systems design, 2024. URL https://arxiv.org/abs/2407.16831.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. Rephrase and respond: Let large language models ask better questions for themselves, 2024. URL https://arxiv.org/abs/2311.04205.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization, 2024. URL `https://arxiv.org/abs/2402.01306`.

Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. *arXiv preprint arXiv:2403.08540*, 2024.

Neel Guha, Mayee F Chen, Trevor Chow, Ishan S Khare, and Christopher Re. Smoothie: Label free language model routing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL `https://arxiv.org/abs/2401.14196`.

Eric Hartford. dolphin-2.2.1-mistral-7b. January 2024.

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

Geoffrey E Hinton et al. *How neural networks learn from experience*. na, 1992.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023a.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise comparison and generative fusion. In *Proceedings of the 61th Annual Meeting of the Association for Computational Linguistics (ACL 2023)*, 2023b.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. Lifelong pretraining: Continually adapting language models to emerging corpora. *arXiv preprint arXiv:2110.08534*, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL `https://arxiv.org/abs/2001.08361`.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need, 2024a. URL `https://arxiv.org/abs/2402.05120`.

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

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 2023.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-star: Learning to interleave thinking and proving, 2024. URL `https://arxiv.org/abs/2407.10040`.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 34(2):550–570, 2021.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. *ArXiv*, 2024.

Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun. Hypermapper: a practical design space exploration framework. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 425–426, 2019. doi: 10.1109/MASCOTS.2019.00053.

Jinjie Ni, Fuzhao Xue, Xiang Yue, Yuntian Deng, Mahir Shah, Kabir Jain, Graham Neubig, and Yang You. Mixeval: Deriving wisdom of the crowd from llm benchmark mixtures, 2024. URL `https://arxiv.org/abs/2406.06565`.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2024. URL `https://arxiv.org/abs/2406.18665`.

OpenAI. Learning to reason with LLMs. `https://openai.com/research/learning-to-reason-with-llms`, September 2024a. Accessed November 13, 2024.

OpenAI. Learning to reason with large language models, 2024b. URL `https://openai.com/index/learning-to-reason-with-llms/`. Accessed: 2024-09-12.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Qwen. Qwen2 technical report. 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. Branch-solve-merge improves large language model evaluation and generation, 2024. URL https://arxiv.org/abs/2310.15123.

Yijia Shao, Yucheng Jiang, Theodore A Kanell, Peter Xu, Omar Khattab, and Monica S Lam. Assisting in writing wikipedia-like articles from scratch with large language models. *arXiv preprint arXiv:2402.14207*, 2024.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012. URL https://arxiv.org/abs/1206.2944.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Llm pruning and distillation in practice: The minitron approach, 2024. URL https://arxiv.org/abs/2408.11796.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Jack Krawczyk, Cosmo Du, Ed Chi, Heng-Tze Cheng, Eric Ni, Purvi Shah, Patrick Kane, Betty Chan, Manaal Faruqui, Aliaksei Severyn, Hanzhao Lin, YaGuang Li, Yong Cheng, Abe Ittycheriah, Mahdis Mahdieh, Mia Chen, Pei Sun, Dustin Tran, Sumit Bagri, Balaji Lakshminarayanan, Jeremiah Liu, Andras Orban, Fabian Güra, Hao Zhou, Xinying Song, Aurelien Boffy, Harish Ganapathy, Steven Zheng, HyunJeong Choe, Ágoston Weisz, Tao Zhu, Yifeng Lu, Siddharth Gopal, Jarrod Kahn, Maciej Kula, Jeff Pitman, Rushin Shah, Emanuel Taropa, Majd Al Merey, Martin Baeuml, Zhifeng Chen, Laurent El Shafey, Yujing Zhang, Olcan Sercinoglu, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, Alexandre Frechette, Charlotte Smith, Laura Culp, Lev Proleev, Yi Luan, Xi Chen, James Lottes, Nathan Schucher, Federico Lebron, Alban Rrustemi, Natalie Clay, Phil Crone, Tomas Kocisky, Jeffrey Zhao, Bartek Perz, Dian Yu, Heidi Howard, Adam Bloniarz, Jack W. Rae, Han Lu, Laurent Sifre, Marcello Maggioni, Fred Alcober, Dan Garrette, Megan Barnes, Shantanu Thakoor, Jacob Austin, Gabriel Barth-Maron, William Wong, Rishabh Joshi, Rahma Chaabouni, Deeni Fatiha, Arun Ahuja, Gaurav Singh Tomar, Evan Senter, Martin Chadwick, Ilya Kornakov, Nithya Attaluri, Iñaki Iturrate, Ruibo Liu, Yunxuan Li, Sarah Cogan, Jeremy Chen, Chao Jia, Chenjie Gu, Qiao Zhang, Jordan Grimstad, Ale Jakse Hartman, Xavier Garcia, Thanumalayan Sankaranarayana Pillai, Jacob Devlin, Michael Laskin, Diego de Las Casas, Dasha Valter, Connie Tao, Lorenzo Blanco, Adrià Puigdomènech Badia, David Reitter, Mianna Chen, Jenny Brennan, Clara Rivera, Sergey Brin, Shariq Iqbal, Gabriela Surita, Jane Labanowski, Abhi Rao, Stephanie Winkler, Emilio Parisotto, Yiming Gu, Kate Olszewska, Ravi Addanki, Antoine Miech, Annie Louis, Denis Teplyashin, Geoff Brown, Elliot Catt, Jan Balaguer, Jackie Xiang, Pidong Wang, Zoe Ashwood, Anton Briukhov, Albert Webson, Sanjay Ganapathy, Smit Sanghavi, Ajay Kannan, Ming-Wei Chang, Axel Stjerngren, Josip Djolonga, Yuting Sun, Ankur Bapna, Matthew Aitchison, Pedram Pejman, Henryk Michalewski, Tianhe Yu, Cindy Wang, Juliette Love, Junwhan Ahn, Dawn Bloxwich, Kehang Han, Peter Humphreys, Thibault Sellam, James Bradbury, Varun Godbole, Sina Samangooei, Bogdan Damoc, Alex Kaskasoli, Sébastien M. R. Arnold, Vijay Vasudevan, Shubham Agrawal, Jason Riesa, Dmitry Lepikhin, Richard Tanburn, Srivatsan Srinivasan, Hyeontaek Lim, Sarah Hodkinson, Pranav Shyam, Johan Ferret, Steven Hand, Ankush Garg, Tom Le Paine, Jian Li, Yujia Li, Minh Giang, Alexander Neitz, Zaheer Abbas, Sarah York, Machel Reid, Elizabeth Cole, Aakanksha Chowdhery, Dipanjan Das, Dominika Rogozińska, Vitaliy Nikolaev, Pablo Sprechmann, Zachary Nado, Lukas Zilka, Flavien Prost, Luheng He, Marianne Monteiro, Gaurav Mishra, Chris Welty, Josh Newlan, Dawei Jia, Miltiadis Allamanis, Clara Huiyi Hu, Raoul de Liedekerke, Justin Gilmer, Carl Saroufim, Shruti Rijhwani, Shaobo Hou, Disha Shrivastava, Anirudh Baddepudi, Alex Goldin, Adnan Ozturel, Albin Cassirer, Yunhan Xu, Daniel Sohn, Devendra Sachan, Reinald Kim Amplayo, Craig Swanson, Dessie Petrova, Shashi Narayan, Arthur Guez, Siddhartha Brahma, Jessica Landon, Miteyan Patel, Ruizhe Zhao, Kevin Villela, Luyu Wang, Wenhao Jia, Matthew Rahtz, Mai Giménez, Legg Yeung, James Keeling, Petko Georgiev, Diana Mincu, Boxi Wu, Salem Haykal, Rachel Saputro, Kiran Vodrahalli, James Qin, Zeynep Cankara, Abhanshu Sharma, Nick Fernando, Will Hawkins, Behnam Neyshabur, Solomon Kim, Adrian Hutter, Priyanka Agrawal, Alex Castro-Ros, George van den Driessche, Tao Wang, Fan Yang, Shuo yiin Chang, Paul Komarek, Ross McIlroy, Mario Lučić, Guodong Zhang, Wael Farhan, Michael Sharman, Paul Natsev, Paul Michel, Yamini Bansal, Siyuan Qiao, Kris Cao, Siamak Shakeri, Christina Butterfield, Justin Chung, Paul Kishan Rubenstein, Shivani Agrawal, Arthur Mensch, Kedar Soparkar, Karel Lenc, Timothy Chung, Aedan Pope, Loren Maggiore, Jackie Kay, Priya Jhakra, Shibo Wang, Joshua Maynez, Mary Phuong, Taylor Tobin, Andrea Tacchetti, Maja Trebacz, Kevin Robinson, Yash Katariya, Sebastian Riedel, Paige Bailey, Kefan Xiao, Nimesh Ghelani, Lora Aroyo, Ambrose Slone, Neil Houlsby, Xuehan Xiong, Zhen Yang, Elena Gribovskaya, Jonas Adler, Mateo Wirth, Lisa Lee, Music Li, Thais Kagohara, Jay Pavagadhi, Sophie Bridgers, Anna Bortsova, Sanjay Ghemawat, Zafarali Ahmed, Tianqi Liu, Richard Powell, Vijay Bolina, Mariko Iinuma, Polina Zablotskaia, James Besley, Da-Woon Chung, Timothy Dozat, Ramona Comanescu, Xiance Si, Jeremy Greer, Guolong Su, Martin Polacek, Raphaël Lopez Kaufman, Simon Tokumine, Hexiang Hu, Elena Buchatskaya, Yingjie Miao, Mohamed Elhawaty, Aditya Siddhant, Nenad Tomasev, Jinwei Xing, Christina Greer, Helen Miller, Shereen Ashraf, Aurko Roy, Zizhao Zhang, Ada Ma, Angelos Filos, Milos Besta, Rory Blevins, Ted Klimenko, Chih-Kuan Yeh, Soravit Changpinyo, Jiaqi Mu, Oscar Chang, Mantas Pajarskas, Carrie Muir, Vered Cohen, Charline Le Lan, Krishna Haridasan, Amit Marathe, Steven Hansen, Sholto Douglas, Rajkumar Samuel, Mingqiu Wang, Sophia Austin, Chang Lan, Jiepu Jiang, Justin Chiu, Jaime Alonso Lorenzo, Lars Lowe Sjösund, Sébastien Cevey, Zach Gleicher, Thi Avrahami, Anudhyan Boral, Hansa Srinivasan, Vittorio Selo, Rhys May, Konstantinos Aisopos, Léonard Hussenot, Livio Baldini Soares, Kate Baumli, Michael B. Chang, Adrià Recasens, Ben Caine, Alexander Pritzel, Filip Pavetic, Fabio Pardo, Anita Gergely, Justin Frye, Vinay Ramasesh, Dan Horgan, Kartikeya Badola, Nora Kassner, Subhrajit Roy, Ethan Dyer, Víctor Campos Campos, Alex Tomala, Yunhao Tang, Dalia El Badawy, Elspeth White, Basil Mustafa, Oran Lang, Abhishek Jindal, Sharad Vikram, Zhitao Gong, Sergi Caelles, Ross Hemsley, Gregory Thornton, Fangxiaoyu Feng, Wojciech Stokowiec, Ce Zheng, Phoebe Thacker, Çağlar Ünlü, Zhishuai Zhang, Mohammad Saleh, James Svensson, Max Bileschi, Piyush Patil, Ankesh Anand, Roman Ring, Katerina Tsihlas, Arpi Vezer, Marco Selvi, Toby Shevlane, Mikel Rodriguez, Tom Kwiatkowski, Samira Daruki, Keran Rong, Allan Dafoe, Nicholas FitzGerald, Keren Gu-Lemberg, Mina Khan, Lisa Anne Hendricks, Marie Pellat, Vladimir Feinberg, James Cobon-Kerr, Tara Sainath, Maribeth Rauh, Sayed Hadi Hashemi, Richard Ives, Yana Hasson, Eric Noland, Yuan Cao, Nathan Byrd, Le Hou, Qingze Wang, Thibault Sottiaux, Michela Paganini, Jean-Baptiste Lespiau, Alexandre Moufarek, Samer Hassan, Kaushik Shivakumar, Joost van Amersfoort, Amol Mandhane, Pratik Joshi, Anirudh Goyal, Matthew Tung, Andrew Brock, Hannah Sheahan, Vedant Misra, Cheng Li, Nemanja Rakićević, Mostafa Dehghani, Fangyu Liu, Sid Mittal, Junhyuk Oh, Seb Noury, Eren Sezener, Fantine Huot, Matthew Lamm, Nicola De Cao, Charlie Chen, Sidharth Mudgal, Romina Stella, Kevin Brooks, Gautam Vasudevan, Chenxi Liu, Mainak Chain, Nivedita Melinkeri, Aaron Cohen, Venus Wang, Kristie Seymore, Sergey Zubkov, Rahul Goel, Summer Yue, Sai Krishnakumaran, Brian Albert, Nate Hurley, Motoki Sano, Anhad Mohananey, Jonah Joughin, Egor Filonov, Tomasz Kępa, Yomna Eldawy, Jiawern Lim, Rahul Rishi, Shirin Badiezadegan, Taylor Bos, Jerry Chang, Sanil Jain, Sri Gayatri Sundara Padmanabhan, Subha Puttagunta, Kalpesh Krishna, Leslie Baker, Norbert Kalb, Vamsi Bedapudi, Adam Kurzrok, Shuntong Lei, Anthony Yu, Oren Litvin, Xiang Zhou, Zhichun Wu, Sam Sobell, Andrea Siciliano, Alan Papir, Robby Neale, Jonas Bragagnolo, Tej Toor, Tina Chen, Valentin Anklin, Feiran Wang, Richie Feng, Milad Gholami, Kevin Ling, Lijuan Liu, Jules Walter, Hamid Moghaddam, Arun Kishore, Jakub Adamek, Tyler Mercado, Jonathan Mallinson, Siddhinita

Wandekar, Stephen Cagle, Eran Ofek, Guillermo Garrido, Clemens Lombriser, Maksim Mukha, Botu Sun, Hafeezul Rahman Mohammad, Josip Matak, Yadi Qian, Vikas Peswani, Pawel Janus, Quan Yuan, Leif Schelin, Oana David, Ankur Garg, Yifan He, Oleksii Duzhyi, Anton Älgmyr, Timothée Lottaz, Qi Li, Vikas Yadav, Luyao Xu, Alex Chinien, Rakesh Shivanna, Aleksandr Chuklin, Josie Li, Carrie Spadine, Travis Wolfe, Kareem Mohamed, Subhabrata Das, Zihang Dai, Kyle He, Daniel von Dincklage, Shyam Upadhyay, Akanksha Maurya, Luyan Chi, Sebastian Krause, Khalid Salama, Pam G Rabinovitch, Pavan Kumar Reddy M, Aarush Selvan, Mikhail Dektiarev, Golnaz Ghiasi, Erdem Guven, Himanshu Gupta, Boyi Liu, Deepak Sharma, Idan Heimlich Shtacher, Shachi Paul, Oscar Akerlund, François-Xavier Aubet, Terry Huang, Chen Zhu, Eric Zhu, Elico Teixeira, Matthew Fritze, Francesco Bertolini, Liana-Eleonora Marinescu, Martin Bölle, Dominik Paulus, Khyatti Gupta, Tejasi Latkar, Max Chang, Jason Sanders, Roopa Wilson, Xuewei Wu, Yi-Xuan Tan, Lam Nguyen Thiet, Tulsee Doshi, Sid Lall, Swaroop Mishra, Wanming Chen, Thang Luong, Seth Benjamin, Jasmine Lee, Ewa Andrejczuk, Dominik Rabiej, Vipul Ranjan, Krzysztof Styrc, Pengcheng Yin, Jon Simon, Malcolm Rose Harriott, Mudit Bansal, Alexei Robsky, Geoff Bacon, David Greene, Daniil Mirylenka, Chen Zhou, Obaid Sarvana, Abhimanyu Goyal, Samuel Andermatt, Patrick Siegler, Ben Horn, Assaf Israel, Francesco Pongetti, Chih-Wei "Louis" Chen, Marco Selvatici, Pedro Silva, Kathie Wang, Jackson Tolins, Kelvin Guu, Roey Yogev, Xiaochen Cai, Alessandro Agostini, Maulik Shah, Hung Nguyen, Noah Ó Donnaile, Sébastien Pereira, Linda Friso, Adam Stambler, Adam Kurzrok, Chenkai Kuang, Yan Romanikhin, Mark Geller, ZJ Yan, Kane Jang, Cheng-Chun Lee, Wojciech Fica, Eric Malmi, Qijun Tan, Dan Banica, Daniel Balle, Ryan Pham, Yanping Huang, Diana Avram, Hongzhi Shi, Jasjot Singh, Chris Hidey, Niharika Ahuja, Pranab Saxena, Dan Dooley, Srividya Pranavi Potharaju, Eileen O'Neill, Anand Gokulchandran, Ryan Foley, Kai Zhao, Mike Dusenberry, Yuan Liu, Pulkit Mehta, Ragha Kotikalapudi, Chalence Safranek-Shrader, Andrew Goodman, Joshua Kessinger, Eran Globen, Prateek Kolhar, Chris Gorgolewski, Ali Ibrahim, Yang Song, Ali Eichenbaum, Thomas Brovelli, Sahitya Potluri, Preethi Lahoti, Cip Baetu, Ali Ghorbani, Charles Chen, Andy Crawford, Shalini Pal, Mukund Sridhar, Petru Gurita, Asier Mujika, Igor Petrovski, Pierre-Louis Cedoz, Chenmei Li, Shiyuan Chen, Niccolò Dal Santo, Siddharth Goyal, Jitesh Punjabi, Karthik Kappaganthu, Chester Kwak, Pallavi LV, Sarmishta Velury, Himadri Choudhury, Jamie Hall, Premal Shah, Ricardo Figueira, Matt Thomas, Minjie Lu, Ting Zhou, Chintu Kumar, Thomas Jurdi, Sharat Chikkerur, Yenai Ma, Adams Yu, Soo Kwak, Victor Ähdel, Sujeevan Rajayogam, Travis Choma, Fei Liu, Aditya Barua, Colin Ji, Ji Ho Park, Vincent Hellendoorn, Alex Bailey, Taylan Bilal, Huanjie Zhou, Mehrdad Khatir, Charles Sutton, Wojciech Rzadkowski, Fiona Macintosh, Konstantin Shagin, Paul Medina, Chen Liang, Jinjing Zhou, Pararth Shah, Yingying Bi, Attila Dankovics, Shipra Banga, Sabine Lehmann, Marissa Bredesen, Zifan Lin, John Eric Hoffmann, Jonathan Lai, Raynald Chung, Kai Yang, Nihal Balani, Arthur Bražinskas, Andrei Sozanschi, Matthew Hayes, Héctor Fernández Alcalde, Peter Makarov, Will Chen, Antonio Stella, Liselotte Snijders, Michael Mandl, Ante Kärrman, Paweł Nowak, Xinyi Wu, Alex Dyck, Krishnan Vaidyanathan, Raghavender R, Jessica Mallet, Mitch Rudominer, Eric Johnston, Sushil Mittal, Akhil Udathu, Janara Christensen, Vishal Verma, Zach Irving, Andreas Santucci, Gamaleldin Elsayed, Elnaz Davoodi, Marin Georgiev, Ian Tenney, Nan Hua, Geoffrey Cideron, Edouard Leurent, Mahmoud Alnahlawi, Ionut Georgescu, Nan Wei, Ivy Zheng, Dylan Scandinaro, Heinrich Jiang, Jasper Snoek, Mukund Sundararajan, Xuezhi Wang, Zack Ontiveros, Itay Karo, Jeremy Cole, Vinu Rajashekhar, Lara Tumeh, Eyal Ben-David, Rishub Jain, Jonathan Uesato, Romina Datta, Oskar Bunyan, Shimu Wu, John Zhang, Piotr Stanczyk, Ye Zhang, David Steiner, Subhajit Naskar, Michael Azzam, Matthew Johnson, Adam Paszke, Chung-Cheng Chiu, Jaume Sanchez Elias, Afroz Mohiuddin, Faizan Muhammad, Jin Miao, Andrew Lee, Nino Vieillard, Jane Park, Jiageng Zhang, Jeff Stanway, Drew Garmon, Abhijit Karmarkar, Zhe Dong, Jong Lee, Aviral Kumar, Luowei Zhou, Jonathan Evens, William Isaac, Geoffrey Irving, Edward Loper, Michael Fink, Isha Arkatkar, Nanxin Chen, Izhak Shafran, Ivan Petrychenko, Zhe Chen, Johnson Jia, Anselm Levskaya, Zhenkai Zhu, Peter Grabowski, Yu Mao, Alberto Magni, Kaisheng Yao, Javier Snaider, Norman Casagrande, Evan Palmer, Paul Suganthan, Alfonso Castaño, Irene Giannoumis, Wooyeol Kim, Mikołaj Rybiński, Ashwin Sreevatsa, Jennifer Prendki, David Soergel, Adrian Goedeckemeyer, Willi Gierke, Mohsen Jafari, Meenu Gaba, Jeremy Wiesner, Diana Gage Wright, Yawen Wei, Harsha Vashisht, Yana Kulizhskaya, Jay Hoover, Maigo Le, Lu Li, Chimezie Iwuanyanwu, Lu Liu, Kevin Ramirez, Andrey Khorlin, Albert Cui, Tian LIN, Marcus Wu, Ricardo Aguilar, Keith Pallo, Abhishek Chakladar, Ginger Perng, Elena Allica Abellan, Mingyang Zhang, Ishita Dasgupta, Nate Kushman, Ivo Penchev, Alena Repina, Xihui Wu, Tom van der Weide, Priya Ponnapalli, Caroline Kaplan, Jiri Simsa, Shuangfeng Li, Olivier Dousse, Fan Yang, Jeff Piper, Nathan Ie, Rama Pasumarthi, Nathan Lintz, Anitha Vijayakumar, Daniel Andor, Pedro Valenzuela, Minnie Lui, Cosmin Paduraru, Daiyi Peng, Katherine Lee, Shuyuan Zhang, Somer Greene, Duc Dung Nguyen, Paula Kurylowicz, Cassidy Hardin, Lucas Dixon, Lili Janzer, Kiam Choo, Ziqiang Feng, Biao Zhang, Achintya Singhal, Dayou Du, Dan McKinnon, Natasha Antropova, Tolga Bolukbasi, Orgad Keller, David Reid, Daniel Finchelstein, Maria Abi Raad, Remi Crocker, Peter Hawkins, Robert Dadashi, Colin Gaffney, Ken Franko, Anna Bulanova, Rémi

Leblond, Shirley Chung, Harry Askham, Luis C. Cobo, Kelvin Xu, Felix Fischer, Jun Xu, Christina Sorokin, Chris Alberti, Chu-Cheng Lin, Colin Evans, Alek Dimitriev, Hannah Forbes, Dylan Banarse, Zora Tung, Mark Omernick, Colton Bishop, Rachel Sterneck, Rohan Jain, Jiawei Xia, Ehsan Amid, Francesco Piccinno, Xingyu Wang, Praseem Banzal, Daniel J. Mankowitz, Alex Polozov, Victoria Krakovna, Sasha Brown, MohammadHossein Bateni, Dennis Duan, Vlad Firoiu, Meghana Thotakuri, Tom Natan, Matthieu Geist, Ser tan Girgin, Hui Li, Jiayu Ye, Ofir Roval, Reiko Tojo, Michael Kwong, James Lee-Thorp, Christopher Yew, Danila Sinopalnikov, Sabela Ramos, John Mellor, Abhishek Sharma, Kathy Wu, David Miller, Nicolas Sonnerat, Denis Vnukov, Rory Greig, Jennifer Beattie, Emily Caveness, Libin Bai, Julian Eisenschlos, Alex Korchemniy, Tomy Tsai, Mimi Jasarevic, Weize Kong, Phuong Dao, Zeyu Zheng, Frederick Liu, Fan Yang, Rui Zhu, Tian Huey Teh, Jason Sanmiya, Evgeny Gladchenko, Nejc Trdin, Daniel Toyama, Evan Rosen, Sasan Tavakkol, Linting Xue, Chen Elkind, Oliver Woodman, John Carpenter, George Papamakarios, Rupert Kemp, Sushant Kafle, Tanya Grunina, Rishika Sinha, Alice Talbert, Diane Wu, Denese Owusu-Afriyie, Cosmo Du, Chloe Thornton, Jordi Pont-Tuset, Pradyumna Narayana, Jing Li, Saaber Fatehi, John Wieting, Omar Ajmeri, Benigno Uria, Yeongil Ko, Laura Knight, Amélie Héliou, Ning Niu, Shane Gu, Chenxi Pang, Yeqing Li, Nir Levine, Ariel Stolovich, Rebeca Santamaria-Fernandez, Sonam Goenka, Wenny Yustalim, Robin Strudel, Ali Elqursh, Charlie Deck, Hyo Lee, Zonglin Li, Kyle Levin, Raphael Hoffmann, Dan Holtmann-Rice, Olivier Bachem, Sho Arora, Christy Koh, Soheil Hassas Yeganeh, Siim Põder, Mukarram Tariq, Yanhua Sun, Lucian Ionita, Mojtaba Seyedhosseini, Pouya Tafti, Zhiyu Liu, Anmol Gulati, Jasmine Liu, Xinyu Ye, Bart Chrzaszcz, Lily Wang, Nikhil Sethi, Tianrun Li, Ben Brown, Shreya Singh, Wei Fan, Aaron Parisi, Joe Stanton, Vinod Koverkathu, Christopher A. Choquette-Choo, Yunjie Li, TJ Lu, Abe Ittycheriah, Prakash Shroff, Mani Varadarajan, Sanaz Bahargam, Rob Willoughby, David Gaddy, Guillaume Desjardins, Marco Cornero, Brona Robenek, Bhavishya Mittal, Ben Albrecht, Ashish Shenoy, Fedor Moiseev, Henrik Jacobsson, Alireza Ghaffarkhah, Morgane Rivière, Alanna Walton, Clément Crepy, Alicia Parrish, Zongwei Zhou, Clement Farabet, Carey Radebaugh, Praveen Srinivasan, Claudia van der Salm, Andreas Fidjeland, Salvatore Scellato, Eri Latorre-Chimoto, Hanna Klimczak-Plucińska, David Bridson, Dario de Cesare, Tom Hudson, Piermaria Mendolicchio, Lexi Walker, Alex Morris, Matthew Mauger, Alexey Guseynov, Alison Reid, Seth Odoom, Lucia Loher, Victor Cotruta, Madhavi Yenugula, Dominik Grewe, Anastasia Petrushkina, Tom Duerig, Antonio Sanchez, Steve Yadlowsky, Amy Shen, Amir Globerson, Lynette Webb, Sahil Dua, Dong Li, Surya Bhupatiraju, Dan Hurt, Haroon Qureshi, Ananth Agarwal, Tomer Shani, Matan Eyal, Anuj Khare, Shreyas Rammohan Belle, Lei Wang, Chetan Tekur, Mihir Sanjay Kale, Jinliang Wei, Ruoxin Sang, Brennan Saeta, Tyler Liechty, Yi Sun, Yao Zhao, Stephan Lee, Pandu Nayak, Doug Fritz, Manish Reddy Vuyyuru, John Aslanides, Nidhi Vyas, Martin Wicke, Xiao Ma, Evgenii Eltyshev, Nina Martin, Hardie Cate, James Manyika, Keyvan Amiri, Yelin Kim, Xi Xiong, Kai Kang, Florian Luisier, Nilesh Tripuraneni, David Madras, Mandy Guo, Austin Waters, Oliver Wang, Joshua Ainslie, Jason Baldridge, Han Zhang, Garima Pruthi, Jakob Bauer, Feng Yang, Riham Mansour, Jason Gelman, Yang Xu, George Polovets, Ji Liu, Honglong Cai, Warren Chen, XiangHai Sheng, Emily Xue, Sherjil Ozair, Christof Angermueller, Xiaowei Li, Anoop Sinha, Weiren Wang, Julia Wiesinger, Emmanouil Koukoumidis, Yuan Tian, Anand Iyer, Madhu Gurumurthy, Mark Goldenson, Parashar Shah, MK Blake, Hongkun Yu, Anthony Urbanowicz, Jennimaria Palomaki, Chrisantha Fernando, Ken Durden, Harsh Mehta, Nikola Momchev, Elahe Rahimtoroghi, Maria Georgaki, Amit Raul, Sebastian Ruder, Morgan Redshaw, Jinhyuk Lee, Denny Zhou, Komal Jalan, Dinghua Li, Blake Hechtman, Parker Schuh, Milad Nasr, Kieran Milan, Vladimir Mikulik, Juliana Franco, Tim Green, Nam Nguyen, Joe Kelley, Aroma Mahendru, Andrea Hu, Joshua Howland, Ben Vargas, Jeffrey Hui, Kshitij Bansal, Vikram Rao, Rakesh Ghiya, Emma Wang, Ke Ye, Jean Michel Sarr, Melanie Moranski Preston, Madeleine Elish, Steve Li, Aakash Kaku, Jigar Gupta, Ice Pasupat, Da-Cheng Juan, Milan Someswar, Tejvi M., Xinyun Chen, Aida Amini, Alex Fabrikant, Eric Chu, Xuanyi Dong, Amruta Muthal, Senaka Buthpitiya, Sarthak Jauhari, Nan Hua, Urvashi Khandelwal, Ayal Hitron, Jie Ren, Larissa Rinaldi, Shahar Drath, Avigail Dabush, Nan-Jiang Jiang, Harshal Godhia, Uli Sachs, Anthony Chen, Yicheng Fan, Hagai Taitelbaum, Hila Noga, Zhuyun Dai, James Wang, Chen Liang, Jenny Hamer, Chun-Sung Ferng, Chenel Elkind, Aviel Atias, Paulina Lee, Vít Listík, Mathias Carlen, Jan van de Kerkhof, Marcin Pikus, Krunoslav Zaher, Paul Müller, Sasha Zykova, Richard Stefanec, Vitaly Gatsko, Christoph Hirnschall, Ashwin Sethi, Xingyu Federico Xu, Chetan Ahuja, Beth Tsai, Anca Stefanoiu, Bo Feng, Keshav Dhandhania, Manish Katyal, Akshay Gupta, Atharva Parulekar, Divya Pitta, Jing Zhao, Vivaan Bhatia, Yashodha Bhavnani, Omar Alhadlaq, Xiaolin Li, Peter Danenberg, Dennis Tu, Alex Pine, Vera Filippova, Abhipso Ghosh, Ben Limonchik, Bhargava Urala, Chaitanya Krishna Lanka, Derik Clive, Yi Sun, Edward Li, Hao Wu, Kevin Hongtongsak, Ianna Li, Kalind Thakkar, Kuanysh Omarov, Kushal Majmundar, Michael Alverson, Michael Kucharski, Mohak Patel, Mudit Jain, Maksim Zabelin, Paolo Pelagatti, Rohan Kohli, Saurabh Kumar, Joseph Kim, Swetha Sankar, Vineet Shah, Lakshmi Ramachandruni, Xiangkai Zeng, Ben Bariach, Laura Weidinger, Tu Vu, Alek Andreev, Antoine He, Kevin Hui, Sheleem Kashem, Amar Subramanya, Sissie Hsiao, Demis Hassabis, Koray Kavukcuoglu, Adam Sadovsky, Quoc Le, Trevor Strohman, Yonghui Wu, Slav Petrov, Jeffrey Dean, and Oriol Vinyals. Gemini: A family of highly capable multimodal models, 2024a. URL https://arxiv.org/abs/2312.11805.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024b. URL https://arxiv.org/abs/2403.08295.

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

Hoang Tran, Chris Glaze, and Braden Hancock. Iterative dpo alignment. Technical report, Snorkel AI, 2023.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clementine Fourrier, Nathan Habib, et al. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*, 2023.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024. URL https://arxiv.org/abs/2406.04692.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CfXh93NDgH.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Yunhua Zhou, Jun Zhan, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance, 2024. URL https://arxiv.org/abs/2403.16952.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. 2024.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2024. URL https://arxiv.org/abs/2410.10762.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017. URL https://arxiv.org/abs/1611.01578.