International Conference on Interdisciplinary Studies in Education, Research and Technology 2024 (ISERT-2024)

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Continuous Architecting for Data Applications

Venkata Sai Chandradeep Telaprolu

chandradeep.msba@gmail.com

---*************************---

Abstract:

Big data architectures have become increasingly preva- lent, with applications such as Twitter using stream processing frameworks like Storm to analyze billions of tweets per minute. However, managing the complex interconnections between components in these systems poses challenges for software architects. This paper introduces OSTIA (On-the-fly Static Topology Inference Analysis), a tool designed to help software architects recover, refactor, and evaluate the architectures of data- intensive applications in real-time. OSTIA provides functionalities such as detecting anti-patterns, verifying architectural safety, and performing static topology analyses. Through case studies in both industrial and open-source contexts, OSTIA demonstrates its utility in supporting continuous architecting and performance improvement of big data systems.

---*************************---

I. INTRODUCTION

In recent years, large-scale neural networks designed for language comprehension and generation have reached exceptional performance across a diverse range of applications [1], [2]. Models such as BERT and T5, structured with either encoder-only or encoder-decoder frameworks, often employ infilling strategies like masked language modeling or span corruption as their pre-training objectives. Typically, these models are then adapted to specific tasks through fine-tuning. Despite achieving state-of-the-art outcomes across a multitude of natural language tasks, they rely heavily on substantial task-specific datasets for fine-tuning. Moreover, updating parts of the model's parameters for each task adds complexity to both the fine-tuning and deployment processes [3].

The introduction of GPT-3 [3] showed that large autoregressive models can handle tasks in a *few-shot* manner, where only a minimal task description and a few examples are provided to guide task completion. These models are trained with a decoder-only structure using a standard left-to-right language modeling approach, where the next token is predicted based on preceding context. This few-shot capability demonstrated impressive results without requiring extensive task-specific datasets or updates to the model's internal parameters.

Since the emergence of GPT-3, newer models have further advanced state-of-the-art performance in autoregressive language modeling. Prominent among these are models like GLaM [4], Gopher [5], Chinchilla [6], Megatron-Turing NLG [7], and LaMDA [8], which achieved high-level few-shot performance across numerous tasks. Like GPT-3, these models are variations of the Transformer architecture. Improvements have generally stemmed from a combination of increasing model size, expanding training data volume, refining datasets,and optimizing computational efficiency through sparsely activated modules [9].

Building on this progress, my work presents a 540-billion parameter, densely activated autoregressive Transformer model trained on a collection of 780 billion high-quality tokens. This milestone was made possible by Pathways [10], a new machine learning framework designed for efficient large-scale model training across extensive accelerator arrays, including TPU v4 Pods. The resulting model, called the Pathways Language Model (PLM), demonstrates exceptional few-shot capabilities across a wide range of tasks, such as natural language processing, coding, and mathematical problem-solving.

The major contributions of this work are highlighted as follows:

- **Scalable efficiency**: I illustrate the extensive use of Pathways—a system that facilitates efficient training of a unified model distributed across thousands of accelerator units. Using this system, I successfully trained a 540-billion parameter model with 6,144 TPU v4 chips, achieving unprecedented scalability and efficiency for a model of this size. Earlier large-scale models either operated on single TPU setups or employed pipeline parallelism across clusters, with a maximum scale of 4,096 TPU v3 chips. My approach delivers superior model and hardware utilization.
- **Ongoing scaling advancements**: I evaluated my model on hundreds of tasks involving language, code, and mathematical reasoning, achieving superior results on most benchmarks. This confirms that scaling improvements for large models have not yet plateaued, with significant gains demonstrated across widely tested benchmarks.
- **Breakthrough capabilities**: I highlight my model's exceptional abilities in complex language comprehension and generation tasks. Specifically, my evaluation includes challenging reasoning tasks that demand multi-step logical or commonsense reasoning. By combining scaling with a technique known as chain-of-thought prompting, I show that simple few-shot evaluation can match or exceed the performance of specialized fine-tuned systems. Additionally, my model achieved standout results on a suite of newly introduced language and reasoning tasks, many of which are considered challenging even for humans. Furthermore, I explore the model's potential to explain complex reasoning chains in a coherent manner.
- **Discontinuous improvements**: To gain deeper insights

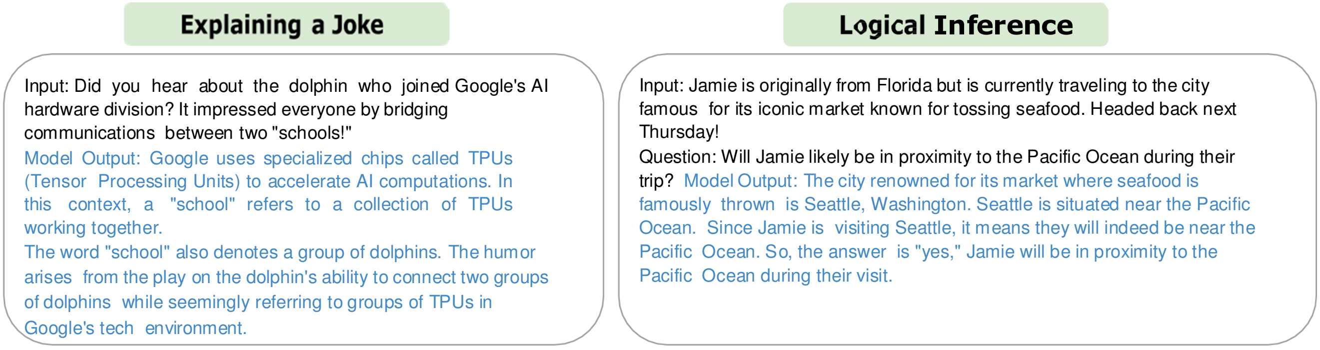


Fig. 1: These examples demonstrate the model's ability to produce detailed explanations using *chain-of-thought prompting*. All outputs were generated with greedy (1-best) decoding of the 540B parameter model. The prompts utilized two-shot exemplars as outlined in Section 1, and additional examples were processed using these exemplars to showcase consistent interpretative responses.

into scaling behaviors, I present findings across three distinct model sizes: 8B, 62B, and 540B parameters. In general, increasing model size from 62B to 540B yields performance gains comparable to those achieved by scaling from 8B to 62B, consistent with the "power law" trend frequently observed in neural network growth [11], [12]. However, in certain tasks, scaling from 62B to 540B results in notable, abrupt accuracy jumps, surpassing the prior rate of improvement. This distinct surge is evident in about 25% of the BIG-bench tasks outlined in Section VI, indicating that scaling large models can unlock new capabilities, which continue to emerge at previously unexplored scales [3].

Multilingual understanding: Previous assessments of large language models have often been constrained within the multilingual space [13]. In this work, I undertake a broader evaluation spanning several multilingual bench- marks, including tasks like machine translation (see Sec- tion VII-B), summarization (refer to Section VII-C), and question answering (outlined in Section VII-D) across diverse languages. Although the training dataset for the 540B model contained a modest share of non-English data (around 22%), the model's few-shot performance closely matches, and sometimes even surpasses, existing state-of-the-art results in non-English summarization and translation [14]. Further investigation is needed to determine how increasing the share of multilingual data might impact the model's effectiveness across both English and multilingual tasks.

Bias and toxicity: I also explored how the model handles

distributional bias and toxicity, leading to several key findings (refer to Section X) [15], [16]. Firstly, I noted improvements in gender and occupation bias performance on the Winogender coreference task as model scale increased, with the 540B model achieving state-of-the- art accuracy in one-shot and few-shot settings. Secondly, analyzing race, religion, and gender co-occurrences inprompt completions exposed potential risks of reinforcing harmful stereotypes by linking certain groups with negative traits [17]. This trend was apparent across different model sizes. Finally, in terms of toxicity, the 62B and 540B models exhibited marginally higher toxicity in generated outputs compared to the 8B model. The content and tone of prompts heavily influenced response toxicity, showing a stronger dependence compared to human- generated text [18]. Moving forward, my work aims to expand these assessments to cover non-English languages and address related risks more comprehensively.

II. Model Design

The architecture of the model follows a Transformer-based framework, specifically designed in a decoder-only layout, where each timestep is constrained to attend only to its own position and those before it. However, several distinct enhancements are integrated into the design:

SwiGLU Activation: For the intermediate MLP layers, I incorporate SwiGLU activations, which are formulated as Swish$(xW) \cdot xV$. This modification outperforms traditional activation functions like ReLU, GeLU, or Swish in terms of model performance. Although this activation requires three matrix multiplications in the MLP instead of two, my experiments have shown that it leads to better results in equivalent compute setups, especially when compared to the more common ReLU- based approach with similarly scaled layers.

- **Parallelized Layer Structure**: Rather than relying on the traditional "sequential" setup for Transformer blocks, I utilize a "parallel" structure. The conventional formulation is:

  $y = x + \text{MLP}(\text{LayerNorm}(x + \text{Attention}(\text{LayerNorm}(x))))$

  In my parallel version, it is expressed as:

  $y = x + \text{MLP}(\text{LayerNorm}(x)) + \text{Attention}(\text{LayerNorm}(x))$

This parallel structure facilitates a speedup of roughly 15% during training at large scales, as it enables the fusion of the input matrix multiplications for MLP and Attention. While I observed some minor quality loss at smaller model scales (8B), no significant degradation occurred for models at the 62B scale and beyond.

Multi-Query Attention Mechanism: Traditional Trans- former models separately project each timestep's input vector into three distinct tensors: "query," "key," and "value," with k heads and an attention size of h. In my design, the key and value projections are shared across attention heads, reducing their dimensionality to $[1, h]$, while keeping the query projection at $[k, h]$. This adjustment improves computational efficiency, especially during autoregressive decoding when only one token is processed at a time, without sacrificing model accuracy or training performance.

Rotary Positional Embeddings (RoPE): I opt for RoPE embeddings instead of absolute or relative positional encodings due to their enhanced performance in handling long input sequences.

Unified Input-Output Embedding Matrices: The same embedding matrices are used for both input and output, a practice observed in earlier models, to streamline the architecture.

Exclusion of Bias Terms: Bias terms are omitted from dense kernels and layer normalization layers, which contributes to more stable training, particularly for large- scale models.

Vocabulary Design: A SentencePiece model with 256k tokens is used to build the vocabulary. This approach ensures that a wide variety of languages can be supported efficiently within the training dataset. The vocabulary is derived from the training corpus, ensuring both lossless and reversible tokenization. Notably, whitespace tokens are preserved, which is especially important for code, and any out-of-vocabulary Unicode characters are split into UTF-8 byte sequences. Numerical values are tokenized by separating digits (e.g., "123.5" is tokenized as "1 2 3 . 5").

A. Model Scale Parameters

In this study, I explore three distinct model sizes: 540 billion parameters, 62 billion parameters, and 8 billion parameters. The number of floating-point operations (FLOPs) per token roughly matches the number of parameters since all models utilize standard dense Transformer configurations. The specific hyperparameters for each model configuration are presented in Table I. These models were trained with the same dataset and vocabulary, with the only variation being the batch size. Additional details on the training process can be found in Sections III and V.

B. Model Card

A comprehensive overview of the model architecture, the training setup, the dataset used, and its intended applications can be found in the Model Card section of the Appendix.

III. Training Data

The pretraining dataset consists of an extensive collection of 780 billion tokens, covering a broad spectrum of natural language contexts. This dataset includes a mix of filtered web pages, books, Wikipedia entries, news articles, source code, and social media exchanges. Web pages were selected based on a "quality score," which was determined using a classifier trained to identify content similar to established high-quality collections. Higher-rated web pages were sampled more frequently, although lower-rated pages were not completely excluded.

Additionally, the dataset contains programming code from publicly available web repositories. These files were filtered by license type, excluding those with copyleft licenses, and focused on 24 common programming languages, including Java, HTML, JavaScript, Python, PHP, C#, XML, C++, and C, making up a total of 196GB of source code. Duplicate files were removed by calculating the Levenshtein distance to ensure diversity in the code corpus.

Table II presents the proportion of each data type included in the final dataset. I also performed data contamination checks and overlap analysis between the training and evaluation datasets to ensure data integrity.

IV. Training Infrastructure

The model training and evaluation were conducted using JAX and T5X, with all models being processed on TPU v4 Pods. The largest model, PaLM 540B, was trained using a network of two TPU v4 Pods interconnected through a data center network (DCN), employing a combination of model and data parallelism. Each Pod consists of 3072 TPU v4 chips spread across 768 hosts, allowing for efficient scaling up to 6144 chips without the use of pipeline parallelism.

Other large-scale training configurations have either relied on single TPU systems or used various parallelism strategies across GPU clusters. Unlike these setups, my architecture efficiently scales without introducing the bottlenecks typically associated with pipeline parallelism.

In pipeline parallelism, training batches are divided into smaller "micro-batches," which can lead to inefficiencies due to pipeline "bubbles" that occur at different computational stages. Additionally, this approach often requires additional memory bandwidth and increases the complexity of software. Instead, I chose to avoid pipelining in the PaLM 540B training while still achieving effective scaling.

Model	Layers	# of Heads	d_{model}	# of Parameters (in billions)	Batch Size
Model 8B	32	16	4096	8.63	256 → 512
Model 62B	64	32	8192	62.50	512 → 1024
Model 540B	118	48	18432	540.35	512 → 1024 → 2048

TABLE I: Summary of the model architectures, including the number of layers, model size d_{model}, attention heads, and attention head size. The feed-forward dimension d_{ff} is consistently set to four times the model size d_{model}, and the attention head size is fixed at 256.

Total dataset size = 780 billion tokens	
Data Source	**Data Proportion**
Multilingual social media dialogues	50%
Multilingual web pages (filtered)	27%
English-language books	13%
Code from Web repositories	5%
Multilingual Wikipedia content	4%
English-language news articles	1%

TABLE II: Distribution of data sources in the training dataset. The multilingual dataset includes text from over 100 languages, with further details on the distribution available in the Appendix.

Within each TPU v4 Pod, the model parameters are distributed across 3072 chips, employing a 12-way model parallelism combined with 256-way fully sharded data parallelism, referred to as "2D finalized." During the forward pass, weights are gathered along the data parallel axis, and activations from each layer are stored. In the backward pass, the remaining activations are recomputed, enhancing throughput for larger batch sizes.

For scaling beyond a single Pod, I employed the Pathways system, which utilizes a client-server model to facilitate two-way data parallelism across the Pods. A single Python client divides the training batch and assigns it to each Pod, which then calculates gradients in parallel. The gradients from all Pods are exchanged before updating the model parameters, ensuring consistency across all Pods for the next training step.

Figure 2 illustrates how Pathways enables parallelism at the Pod level. The client creates a distributed dataflow program for JAX/XLA, which assigns computation tasks to the TPU Pods. The program includes components for local computations within each Pod, cross-Pod gradient transfers, and optimizer updates. By leveraging asynchronous scheduling and optimizing data transfers through sharded execution, this system ensures efficient scaling even when using thousands of chips.

Transferring gradients between Pods poses a significant challenge, as it requires efficient data communication between 6144 TPU chips distributed over 1536 hosts. Each TPU core only exchanges gradients corresponding to its specific parameters, generating bursts of network traffic. Optimization techniques such as data chunking and advanced routing strategies maximize throughput, enabling near-perfect scaling efficiency across Pods.

A. Training Efficiency

Traditional metrics for accelerator efficiency, such as hardware FLOPs utilization (HFU), have some limitations due to system-specific factors and implementation decisions. For instance, rematerialization, a technique that exchanges memory usage for computation, can increase hardware FLOPs but does not necessarily improve overall training efficiency. Thus, HFU does not provide a consistent basis for comparing training efficiency across different systems.

We propose a new metric called Model FLOPs Utilization (MFU), which measures the actual throughput (tokens per second) relative to the theoretical maximum throughput of a system. This metric considers only the operations needed for the forward and backward passes, excluding rematerialization. MFU thus offers a standardized method for comparing the training efficiency across different models and hardware setups.

PaLM 540B achieves an MFU of 46.2%, surpassing previous models by a significant margin. The use of rematerialization allows for larger batch sizes and accelerated training, resulting in a total hardware FLOPs utilization of 57.8%.

V. Training Configuration

The model follows a standard Transformer training approach, with the following specifications:

- **Weight Initialization** – The kernel weights are initialized using fan-in variance scaling, while input embeddings are initialized using a standard normal distribution.
- **Optimizer** – The Adafactor optimizer is employed with parameter scaling, essentially equivalent to Adam, but without adjusting the learning rate for embeddings and layer normalization parameters.
- **Hyperparameters** – The initial learning rate is set to 10^{-2}, which decays according to a $1/\sqrt{k}$ schedule. Momentum is fixed at 0.9, and gradient clipping is applied using a global norm value of 1.0.
- **Loss Function** – The model is trained using the conventional language modeling loss function, supplemented by a small auxiliary loss to ensure training stability.

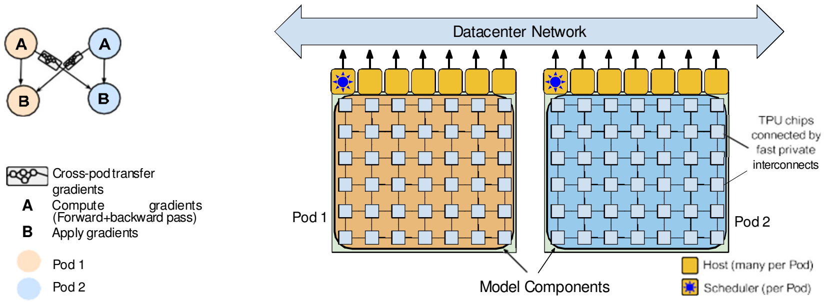


Fig. 2: The Pathways framework enables training scalability across two TPU v4 Pods, utilizing two-way data parallelism at the Pod level.

TABLE III: Model FLOPs efficiency for PaLM and other large-scale models. PaLM demonstrates significant MFU performance owing to several optimization strategies.

Model	Parameters (B)	Accelerators	FLOPS Utilization
GPT-3	175	V100	21.3%
Gopher	280	4096 TPU v3	32.5%
Megatron-Turing NLG	530	2240 A100	30.2%
PaLM	540	6144 TPU v4	46.2%

- **Sequence Length** – A maximum sequence length of 2048 tokens is used, with no padding, though some input examples may be split.
- **Batch Size** – The batch size starts at 512 and progressively increases to 2048 throughout training, balancing sample efficiency with gradient estimation.
- **Bitwise Determinism** – The training process is fully reproducible, utilizing a deterministic framework and data pipeline.
- **Dropout** – No dropout is applied during the initial training phase, but a rate of 0.1 is applied during fine-tuning.

A. Training Instability

Training the largest model sometimes resulted in occasional loss spikes, occurring 20 times during training despite the application of gradient clipping. These spikes were unpredictable and did not occur in smaller models. Upon further investigation, it was determined that these spikes were not caused by "bad data." To address this, I implemented a strategy of restarting training from a checkpoint before the spike and skipping a few data batches. This approach prevented the spikes from recurring at the same points. I plan to explore more effective methods for handling such training instability in future iterations.

1) Fine-tuning: We conduct fine-tuning experiments on the PaLM model utilizing the SuperGLUE benchmark. The fine-tuning process is performed with a learning rate of $5 \cdot 10^{-5}$ using the Adafactor optimizer and a batch size of 32. Typically, PaLM reaches convergence in under 15×10^3 training steps.

Table V shows the validation outcomes from fine-tuning on a balanced combination of SuperGLUE tasks. When compared to leading models like T5-11B and ST-MoE-32B, PaLM achieves near-top performance, with results approaching the highest-performing models. Both T5-11B and ST-MoE-32B utilize encoder-decoder architectures trained with the span corruption method, which has been demonstrated to surpass autoregressive decoder-only models in classification task fine- tuning when training costs are equivalent. These findings highlight the significant impact of model scale in narrowing the performance gap. Additionally, Table VI illustrates a clear performance difference between few-shot and fine-tuned outcomes. Finally, Table VII presents the test set results from the SuperGLUE leaderboard, where PaLM surpasses the leading decoder-only autoregressive language model by a considerable margin.

VI. BIG-BENCH EVALUATION

BIG-bench is a benchmark comprising over 150 diverse tasks designed to evaluate large language models, covering domains like reasoning, translation, and mathematics. In this section, I report the few-shot evaluation results of the PaLM model family on the textual tasks of BIG-bench, excluding programmatic tasks.

The benchmark compares human and model performance. Human workers were provided with task descriptions and examples, with access to external tools like search engines. The "best" human performance represents the highest score per example, while the "average" human score is the mean of all submissions.

The results indicate that PaLM outperforms previous models such as GPT-3, Gopher, and Chinchilla on 58 common tasks, achieving a higher score than the average human performance. PaLM 540B in a 5-shot setup surpasses the prior state-of-

Task	0-shot		1-shot		Few-shot	
	Previous SOTA	PaLMshort 540B	Previous SOTA	PaLMshort 540B	Previous SOTA	PaLMshort 540B
TriviaQA (EM)	71.3[a]	state-of-the-art76.9	75.8[a]	state-of-the-art81.4	75.8[a] few-shot1	state-of-the-art81.4 few-shot1
Natural Questions (EM)	state-of-the-art24.7[a]	21.2	26.3[a]	state-of-the-art29.3	32.5[a] few-shot1	state-of-the-art39.6 few-shot64
Web Questions (EM)	state-of-the-art19.0[a]	10.6	state-of-the-art25.3[b]	22.6	41.1[b] few-shot64	state-of-the-art43.5 few-shot64

TABLE IV: Performance of PaLM 540B across 29 NLP benchmarks. Superscripts indicate prior results from previous works: [a]GLaM 62B/64E, [b]GPT-3 175B. The results are presented for 0-shot, 1-shot, and few-shot scenarios, with shot numbers provided in parentheses for few-shot settings.

TABLE V: Validation performance on the SuperGLUE development set, comparing with T5-11B and ST-MoE-32B. The scores presented are the peak validation scores for each task.

Model	Avg	BoolQ	CB	CoPA	MultiRC	Record	RTE	WiC	WSC
T5-11B	89.9	90.8	94.9/96.4	98.0	87.4/66.1	93.8/93.2	93.9	77.3	96.2
ST-MoE-32B	93.2	93.1	100	100	90.4/69.9	95.0/95.6	95.7	81.0	100
PaLMshort 540B (*fine-tuned*)	92.6	92.2	100	100	90.1/69.2	94.0/94.6	95.7	78.8	100

Model	BoolQ	CB	CoPA	MultiRC	Record	RTE	WiC	WSC
Few-shot	89.1	89.3	95	86.3/-	92.9/-	81.2	64.6	89.5
Fine-tuned	92.2	100/100	100	90.1/69.2	94.0/94.6	95.7	78.8	100

TABLE VI: C omparing the performance of PaLMshort-540B in few-shot and fine-tuned settings on SuperGLUE.

Model	Avg	BoolQ	CB	CoPA	MultiRC	Record	RTE	WiC	WSC
ST-MoE-32B	state-of-the-art91.2	state-of-the-art92.4	state-of-the-art96.9/state-of-the-art98.0	state-of-the-art99.2	state-of-the-art89.6/state-of-the-art65.8	state-of-the-art95.1/state-of-the-art94.4	93.5	state-of-the-art77.7	state-of-the-art96.6
Best Decoder-only LM	71.8	76.4	52.0/75.6	92.0	75.4/30.5	91.1/90.2	69.0	49.4	80.1
PaLMshort 540B (*fine-tuned*)	90.4	91.9	94.4/96.0	99.0	88.7/63.6	94.2/93.3	state-of-the-art95.9	77.4	95.9

TABLE VII: Results on the SuperGLUE test set from the leaderboard, comparing PaLM with state-of-the-art encoder-decoder models and the best decoder-only language model.

the-art on 44 of these tasks. Additionally, as the model scale increases, performance improves, suggesting that further scaling could lead to even better results.

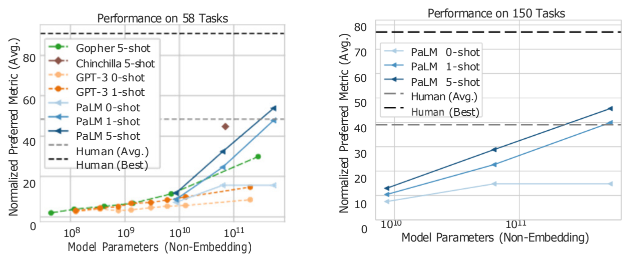


Fig. 3: PaLM's performance on BIG-bench: (left) Comparison with GPT-3, Gopher, and Chinchilla across 58 common tasks. (right) PaLM's results on 150 tasks, with normalized scores for multiple-choice tasks.

VII. Key Tasks where PaLM Demonstrated Significant Performance

Key tasks where PaLM demonstrated notable performance improvements include:

- **Goal-Step Relationship:** Determining the correct se- quence of actions required to achieve a specific goal.

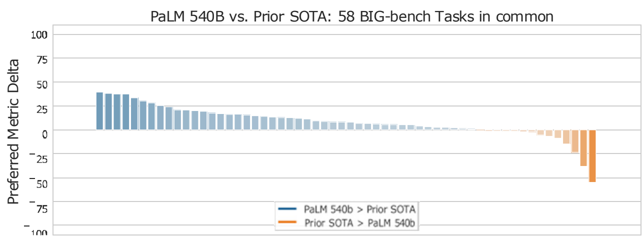


Fig. 4: PaLM 540B 5-shot performance compared to the prior state-of-the-art across 58 common tasks. The blue regions indicate where PaLM outperforms previous models.

- **Logical Inference:** Analyzing a text to draw logical conclusions.
- **Proverbs:** Associating a situation with the most fitting proverb.
- **Logical Sequence:** Arranging events or items in a logically correct order.
- **Navigation:** Following instructions to identify a destina- tion.
- **Mathematical Induction:** Using induction principles to deduce conclusions from premises.

As illustrated in Figure 5, certain tasks like `goal_step_wikihow` and `logical_args` exhibit log-linear scaling, with the PaLM 540B model approaching

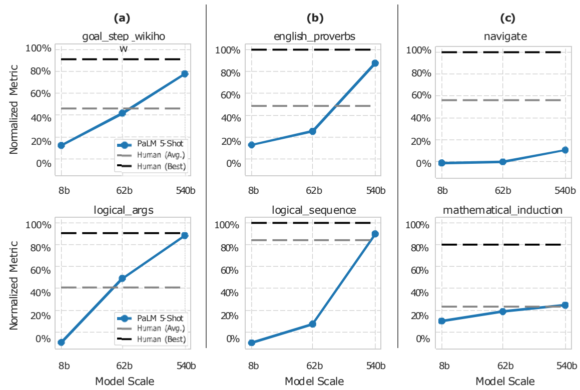


Fig. 5: 5-shot evaluations on six tasks with interesting scaling behaviors. (a) shows log-linear improvements, (b) shows discontinuous improvements, and (c) shows flat improvements.

human-level accuracy. In contrast, tasks such as `english_proverbs` and `logical_sequence` display non-linear progress, where scaling from 62B to 540B results in a substantial leap, more so than from 8B to 62B. This suggests that some capabilities only manifest when the model reaches a certain size. For instance, the `english_proverbs` task shows a large accuracy improvement from 25% to 87% when increasing from PaLM 62B to 540B, indicating the model's growing ability to handle abstract reasoning.

These non-linear improvements are further exemplified by the `logical_sequence` task. The PaLM 540B model achieves 87% accuracy, which exceeds the linear projection (37%) based on smaller models. This sharp increase (+50%) suggests that, for some complex tasks, scaling up the model significantly impacts performance. Across 150 tasks, 25% demonstrate a discontinuous improvement of over +10%, while 15% show a greater-than +20% improvement, highlighting the role of model size in tackling more difficult challenges.

However, scaling doesn't always guarantee improvements in every task. For example, tasks like `navigate` and `mathematical_induction` see only slight gains when moving from the 62B to the 540B model, and performance still falls short of human-level accuracy. In the case of `mathematical_induction`, errors in premise assumptions can hinder the model's performance, even for larger versions.

In Figure 6, I compare PaLM 540B's performance to that of humans across 150 tasks. While PaLM 540B outperforms the average human score in most cases, it still trails human performance on 35% of the tasks, suggesting there is room for further improvement.

PaLM 540B excels in several tasks where it surpasses human averages, particularly those involving multilingual capabilities like `persian_idioms` and

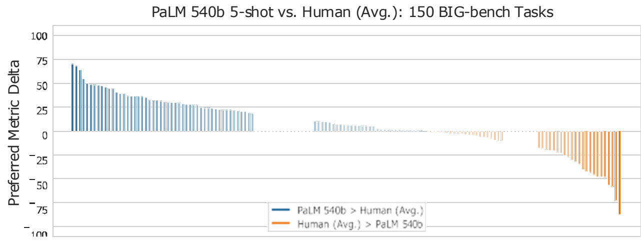


Fig. 6: Distribution of score differences between PaLM 540B and the average human performance across 150 tasks. Positive values (blue) indicate PaLM outperforms humans; negative values (orange) indicate the opposite.

`swedish_to_german_proverbs`. Additionally, tasks requiring memorization and advanced language skills, such as `periodic_elements` and `logical_args`, also see PaLM outperforming human-level performance. A particular task, `cause_and_effect`, tests PaLM's ability to deduce causal relationships from sentences. It is presented in two versions:

A. *Cause and Effect Task Versions*

- **cause_and_effect (one_sentence_no_prompt):** The events are combined into a single sentence with different orderings, and PaLM ranks the likelihood of each sentence.
- **cause_and_effect (two_sentence):** PaLM selects the event that caused the other based on two separate sentences.

The PaLM 540B model performs exceptionally well in both versions, especially in the two-sentence version, achieving over 90% accuracy. This showcases how scaling up the model unlocks increasingly sophisticated language modeling capabilities.

Finally, Figure 7 provides results from BIG-bench Lite, a subset of 24 tasks. While PaLM 540B outperforms all other models, it still falls short of human performance on most tasks, indicating potential for further development. Task `t24` proves challenging for both the model and humans.

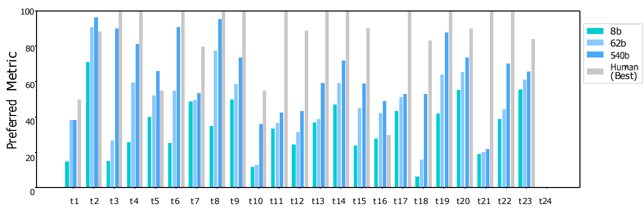


Fig. 7: Performance on 24 tasks in BIG-bench Lite using 5-shot evaluations. PaLM 540B outperforms all other models, but it still lags behind human performance on most tasks, suggesting potential for improvement. Task t24 presents difficulty for both the model and humans. See the appendix for detailed task names and data.

To validate the results, I ensured no data leakage from BIG-bench during model training. The dataset contained unique identifiers (such as a canary string) that were verified to be absent in PaLM's training data. Furthermore, the dataset was not publicly available during training, and many tasks were specifically designed for BIG-bench, ensuring no gold labels influenced the model's predictions in the areas where it excelled.

a) DeepFix Code Correction: The PaLM-Coder 540B model has achieved remarkable results in the DeepFix code correction task, with a compilation success rate of 82.1%, surpassing the 71.7% achieved by previous methods. Figures 8 and 9 illustrate instances where PaLM-Coder successfully predicted the correct code. To generate the prompts, I manually created two pairs of flawed and corrected C programs that included a variety of common coding mistakes, without further refining the prompt examples. I then formatted the broken code using a code formatter before passing it to the model, which in turn generated the corrected code.

For evaluating code repairs, it's important to determine the extent of code modifications—ideally, I aim to change only a minimal portion of the erroneous code. Table VIII provides an analysis based on various metrics to define what constitutes "small" edits.[1] Interestingly, PaLM produces the smallest edits overall, while PaLM-Coder shows the highest success rate when only considering small normalized edit distances. On the other hand, Codex (Davinci) performs better when counting the number of lines modified. In simpler terms, PaLM-Coder tends to make subtle changes spread across more lines compared to Codex, which often makes more drastic changes. This is evident in the predictions, where PaLM-Coder is more likely to make minor stylistic adjustments (such as changing `i = i + 1` to `i++` and transforming `int a;\n int b;` into `int a, b;`) compared to Codex, which is more likely to alter entire lines.

We also explored how PaLM-Coder's predictions would change with a slightly modified prompt. I ran PaLM-Coder using the same pair of prompt examples, except this time, one of the variable declarations was split into two lines in both the flawed and fixed versions. This change aimed to reduce PaLM-Coder's tendency to combine variable declarations into a single line. When using the new prompts, I observed improvements across all metrics in Table VIII, with particularly notable increases in the metrics based on the number of lines changed compared to those based on normalized edit distances. For instance, the overall compilation success rate rose from 82.1% to 83.4%, and the rate of success for edits involving no more than 5 lines increased from 66.8% to 70.2%. Qualitatively, I found that with the modified prompts, PaLM-Coder was less likely to combine variable declarations as expected. While this was just a single experiment, the results were promising, showing that a simple adjustment to the prompts led to better model predictions in the desired direction.

b) Discussion: Deploying language model-based systems in software development introduces challenges, particularly around code correctness and potential hidden bugs. This concern becomes more pronounced with the threat of data poisoning attacks targeting code autocompletion systems and the tendency of language models to propagate existing errors found in prompts. Existing strategies to address these issues remain suboptimal. Developers can manually review suggested code, but this method may not always catch subtle bugs. Furthermore, using test suites to validate suggestions provides some assurance, but small-scale testing does not guarantee functional correctness. While I use small sets of tests to gauge functional correctness for this evaluation, it is acknowledged that more rigorous testing is necessary for a complete assessment. This issue is especially prominent in program repair literature, where many patches suggested by previous systems have proven incorrect due to limited testing coverage.

Ensuring functional accuracy is only one aspect of code quality. Suggested changes by language models must also be maintainable, robust, efficient, and secure. The DeepFix example highlights a potential flaw with PaLM-Coder predictions—while the corrections enable compilation, they may not adhere to robust input formatting or size assumptions. The dataset for DeepFix consists of student-submitted solutions to exercises in a C programming course, where such assumptions were permitted. However, in broader, more general use cases, these suggestions might be inappropriate. Helping developers comprehend and trust the proposed code modifications remains a significant challenge. There has been some prior research on readability and security, but further work is required as code generation models continue to evolve.

c) DeepFix Code Repair: The PaLM-Coder 540B model shows high effectiveness when applied to DeepFix tasks, achieving a compilation success rate of 82.1%. Figures 10 and 11 illustrate some successful predictions by PaLM-Coder. The model was provided with input-output pairs consisting of broken and corrected C code samples that included typical mistakes. After processing the faulty code, the model predicted corrected outputs.

We evaluated the scope of changes made by the model, ideally seeking minimal modifications necessary for a functional fix.

[0] Tasks in BIG-bench Lite include t1: auto_debugging, t2: bbq_lite_json, t3: code_line_description, t4: conceptual_combinations, t5: conlang_translation, t6: emoji_movie, t7: formal_fallacies_syllogisms_negation, t8: hindu_knowledge, t9: known_unknowns, t10: language_identification, t11: logic_grid_puzzle, t12: logical_deduction, t13: misconceptions_russian, t14: novel_concepts, t15: operators, t16: parsinlu_reading_comprehension, t17: play_dialog_same_or_different, t18: repeat_copy_logic, t19: strange_stories, t20: strategyqa, t21: symbol_interpretation, t22: vitaminc_fact_verification, t23: winowhy, t24: linguistics_puzzles.

[1] Previous methods for DeepFix typically modified no more than 5 lines of code. However, in my approach, I first format the broken code, which often increases the number of lines. Additionally, some errors require modifications beyond 5 lines for correction. The DeepFix dataset does not include ground truth fixes or input-output examples to assess repair quality beyond simple compilation success.


```c
#include <stdio.h>
int main() {
    int a[i];
    int i;
    scanf("%d", &n);
    scanf("%d", &a[i]);
    int count = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (a[i] == a[j])
                count = count + 1;
        }
    }
    if (count == a[i])
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int a[100], i, n, count = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i] == a[j])
                count = count + 1;
        }
    }
    if (count == n)
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

Fig. 8: An illustration of a DeepFix task where the original erroneous code is on the left, and the prediction from the PaLM-Coder 540B model is on the right. The predicted code corrects all compilation issues (e.g., undeclared variables) and enhances the code's structure (e.g., grouping variable declarations together) and logic (e.g., reading numbers into the array `a` inside the loop and not misusing index `i` outside of it).

```c
#include <stdio.h>
int main() {
    int n;
    scanf("%d", &n);
    int i, num[400];
    for (i = 0; i < 2 * n; i++)
        scanf("%d", &num[i]);
    int min = 400;
    for (i = 0; i < n; i++) {
        for (int j = 0; j < (2 * n - 1); j++) {
            if (num[i] == num[j])
                int t;
            t = (j - i);
            if (t <= min)
                min = t;
            t = 0;
        }
    }
    printf("%d", min);
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int n, i, min = 400;
    int num[400];
    scanf("%d", &n);
    for (i = 0; i < 2 * n; i++)
        scanf("%d", &num[i]);
    for (i = 0; i < n; i++) {
        for (int j = 0; j < (2 * n - 1); j++) {
            if (num[i] == num[j]) {
                int t = (j - i);
                if (t <= min)
                    min = t;
            }
        }
    }
    printf("%d", min);
    return 0;
}
```

Fig. 9: Another example from DeepFix. In this case, the predicted code resolves the compilation error caused by missing braces around the `if` block, which previously caused a scope issue with variable `t`, and it also makes several improvements, such as declaring variables in one statement and removing the ineffective line `t = 0;`.

Relative to other models, PaLM tends to apply more granular edits across several lines, while Codex often alters larger code blocks more drastically. Predictions from PaLM-Coder frequently introduce stylistic changes, such as consolidating variable declarations and optimizing expressions.

By adjusting the prompt to discourage combined variable declarations, I observed an improvement in the compile rate from 82.1% to 83.4%. This modification also led to more focused code corrections and fewer line alterations.

d) Discussion: Integrating language models into software development workflows poses challenges around ensuring code correctness and avoiding the introduction of new bugs. Developers are encouraged to thoroughly vet model-generated code, as even sophisticated models like PaLM-Coder may yield functional code with potential security or performance concerns. Rigorous testing remains critical for evaluating code quality. For DeepFix, although the model enhances compilation rates, generated solutions may still harbor issues, such as insecure assumptions about input. Ongoing research on code readability, security, and reliability is vital as these models continue to evolve.

B. Translation

Machine translation involves converting text from one human language to another while preserving meaning, semantics, and tone. Models like GPT-3 have shown capabilities in translation even without explicit parallel text training. The results are often impressive for translations into English, though translations away from English are less robust. This section examines PaLM's translation abilities across different language pairs and contexts, noting that while PaLM wasn't explicitly trained on parallel text data, some of this data may appear naturally in

	Pretraining Only		Code-Specific Finetuning	
	LaMDA 137B	PaLMshort 540B	Davinci Codex	PaLM-Coder 540B
Compilation Rate (all edit sizes)	4.3	73.7	81.1	**82.1**
Normalized Edit Distance ≤ 0.10	1.7 (1.9)	68.0 (**93.1**)	70.5 (86.7)	**71.7** (87.5)
Normalized Edit Distance ≤ 0.15	2.1 (2.7)	71.0 (**96.5**)	74.7 (91.7)	**77.0** (94.0)
Normalized Edit Distance ≤ 0.20	2.5 (3.4)	72.4 (**98.1**)	77.1 (94.7)	**79.4** (96.7)
Lines Changed ≤ 5	1.7 (2.1)	63.8 (**87.8**)	**67.9** (83.3)	66.8 (81.1)
Lines Changed ≤ 6	2.1 (3.1)	66.6 (**91.4**)	**71.5** (87.9)	70.1 (85.5)
Lines Changed ≤ 7	2.5 (4.1)	68.4 (**93.7**)	**73.9** (90.9)	72.9 (88.7)

TABLE VIII: The success rates for the DeepFix tasks are represented as percentages, where success is defined by whether the predicted code compiles and satisfies the criteria for a "minor modification." The values in parentheses indicate the proportion of predictions that fall within the "minor modification" category. The "Normalized Edit Distance" is computed using the formula $LevenshteinDistance(x, y)/\max\{len(x), len(y)\}$, where x and y are the strings being compared. The "Lines Changed" metric counts the total number of lines that were added, deleted, or altered, excluding any changes made to indentation.

	Pretraining Only		Finetuned on Code	
	LaMDA 137B	PaLMshort 540B	Davinci Codex	PaLM-Coder 540B
Compilation Rate (all edit sizes)	4.3	73.7	81.1	**82.1**
Normalized Edit Distance ≤ 0.10	1.7 (1.9)	68.0 (**93.1**)	70.5 (86.7)	**71.7** (87.5)
Normalized Edit Distance ≤ 0.15	2.1 (2.7)	71.0 (**96.5**)	74.7 (91.7)	**77.0** (94.0)
Normalized Edit Distance ≤ 0.20	2.5 (3.4)	72.4 (**98.1**)	77.1 (94.7)	**79.4** (96.7)
Lines Changed ≤ 5	1.7 (2.1)	63.8 (**87.8**)	**67.9** (83.3)	66.8 (81.1)
Lines Changed ≤ 6	2.1 (3.1)	66.6 (**91.4**)	**71.5** (87.9)	70.1 (85.5)
Lines Changed ≤ 7	2.5 (4.1)	68.4 (**93.7**)	**73.9** (90.9)	72.9 (88.7)

TABLE IX: Success rates on DeepFix tasks for various models, where a success involves the code compiling with minor edits. Percentages in parentheses represent instances of small edits. The Normalized Edit Distance is calculated via the Levenshtein distance metric applied to code strings, while "Lines Changed" tracks edits, insertions, and deletions but excludes indentation alterations.

the corpus. I focus on language pairs with test and development sets from WMT and categorize these into three groups:

- **English-Centric Language Pairs** — These include language pairs where English serves as the source or target. Resource levels vary based on available parallel data. I use high-resource (WMT'14 English-French), mid-resource (WMT'16 English-German), and low-resource (WMT'16 English-Romanian) pairs for analysis.
- **Direct Language Pairs** — These pairs are directly translated between two languages without English acting as a pivot, reflecting the growing need for broader language pair coverage. For this evaluation, I focus on WMT'19 French-German translations.
- **Extremely-Low Resource Language Pairs** — This involves scenarios with minimal monolingual data for one language, e.g., Kazakh. French and German have billions of tokens in training, while Kazakh has around 134 million tokens. I use WMT'19 English-Kazakh to evaluate this category.

a) Performance Assessment on English-focused Translation Tasks: Our initial evaluation of PaLM focuses on typical English-centered translation tasks used for benchmarking large-scale language models. I explore its capabilities across zero-shot, one-shot, and few-shot settings and contrast its performance with competing models under similar conditions. Results, as summarized in Table X and depicted in Figure 12- left, indicate that PaLM consistently outperforms established baselines, at times by as much as a 13-point BLEU score margin. I also juxtapose these findings with the best available results for specialized translation models on the same language pairs. Although PaLM surpasses some supervised systems, such as in German-English and Romanian-English tasks, I note that some baselines may not fully reflect current WMT task standards.

Figure 12-right visualizes PaLM's zero-shot performance as model size scales from 8B to 62B and 540B parameters. Significant BLEU score increases, notably 13 points for English-German and 17 points for English-French, are observed when moving from 62B to 540B configurations, suggesting scale-driven nonlinear improvements.

1) Key Observations and Insights: Our analysis leads to the following conclusions:

- **Translation into English tends to be superior.** This trend aligns with the performance of many English-centric models and remains evident in PaLM's results. Enhanced multilingual training might help address this discrepancy.
- **Language-specific prompts add significant value.** In

```c
#include <stdio.h>
int main() {
    int a[i];
    int i;
    scanf("%d", &n);
    scanf("%d", &a[i]);
    int count = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (a[i] == a[j])
                count = count + 1;
        }
    }
    if (count == a[i])
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int a[100], i, n, count = 0;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (a[i] == a[j])
                count = count + 1;
        }
    }
    if (count == n)
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

Fig. 10: An example of a DeepFix task, with the original faulty code displayed on the left and the corrected code predicted by the PaLM-Coder 540B model on the right. The corrections resolve all compilation issues and enhance the code's structure and logic.

```c
#include <stdio.h>
int main() {
    int n;

    scanf("%d", &n);
    int i, num[400];
    for (i = 0; i < 2 * n; i++)
        scanf("%d", &num[i]);
    int min = 400;
    for (i = 0; i < n; i++) {
        for (int j = 0; j < (2 * n - 1); j++) {
            if (num[i] == num[j])
                int t;
            t = (j - i);
            if (t <= min)
                min = t;
            t = 0;
        }
    }
    printf("%d", min);
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int n, i, min = 400;
    int num[400];
    scanf("%d", &n);

    for (i = 0; i < 2 * n; i++)
        scanf("%d", &num[i]);

    for (i = 0; i < n; i++) {
        for (int j = 0; j < (2 * n - 1); j++) {
            if (num[i] == num[j]) {
                int t = (j - i);
                if (t <= min)
                    min = t;
            }

        }
    }
    printf("%d", min);
    return 0;
}
```

Fig. 11: Another DeepFix example. The model-generated prediction eliminates compilation errors and refines the code through improvements, such as combining variable declarations and removing redundant elements.

numerous cases, PaLM's zero-shot prompts, which provide language labels, outperform single example-driven prompts used in one- or few-shot contexts, consistent with past research.

- **General-purpose models rival specialists.** Despite PaLM's larger parameter count, its results are competitive with or superior to much smaller dedicated translation models, emphasizing its versatility. However, it remains open whether specialized or generalist models offer greater benefits in data-rich translation scenarios.

C. *Multilingual Text Generation Tasks*

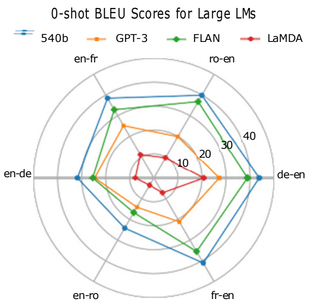
Text generation tasks involve creating coherent and contextually accurate text based on structured or unstructured inputs, such as tables, documents, or other forms of data. The generated text aims to fulfill specific communication objectives, such as summarizing content or converting data into descriptive language. Few-shot performance in conditional generation remains underexplored for models of similar scale. Evaluations of generation tasks often focus on generative question-answering or multiple-choice tests, which typically do not require comprehensive sentence or paragraph creation. Competing large models such as GPT-3, GLaM, Gopher, LaMDA, and Megatron-Turing NLG have not reported few-shot or finetuning results on these specific conditional generation tasks, whether for English or other languages.

This work provides a novel benchmark assessment for few-shot performance of large language models in conditional generation tasks. For reference, comparisons are made against LaMDA 137B, which did not provide results for these benchmarks but was included in my trials.

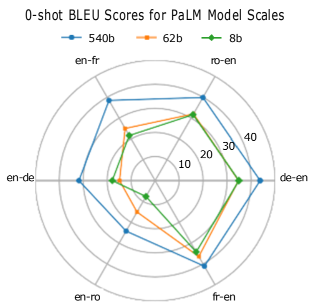
Previous SOTA (State of the Art) approaches for finetuning predominantly used encoder-decoder models like T5, mT5, or BART, trained on tasks involving masking or infilling (e.g., span corruption). These models, with sizes ranging from 130M to 13B parameters, are much smaller compared to PaLM.

		Zero-shot		One-shot		Few-shot		Supervised
Source	Target	Previous SOTA	PaLM 540B	Previous SOTA	PaLM 540B	Previous SOTA	PaLM 540B	Tuned SOTA
en	fr	32.9	**38.5**	28.3	**37.5**	33.9	**44.0**	45.6
en	de	25.4	**31.8**	26.2	**31.8**	26.8	**37.4**	41.2
en	ro	16.7	**24.2**	20.6	**28.2**	20.5	**28.7**	33.4
fr	en	35.5	**41.1**	33.7	**37.4**	38.0	**42.8**	45.4
de	en	38.9	**43.8**	30.4	**43.9**	40.6	**47.5**	41.2
ro	en	36.8	**39.9**	38.6	**42.1**	37.3	**43.8**	39.1

TABLE X: BLEU score comparison for translation tasks involving common WMT language pairs. Best performance across zero- and few-shot scenarios is emphasized in bold, while the overall highest score is underlined. PaLM's few-shot results use a 5-example setup. The zero-shot prompt provides source and target language context, and one-shot and few-shot scenarios derive the language from examples, explaining PaLM's robust zero-shot results.



(a)



(b)

Fig. 12: Comparative analysis of PaLM on zero-shot translation benchmarks. (left) Performance versus earlier large-scale language models. (right) Impact of scaling PaLM's size.

As described in Section V-A1, encoder-decoder models are often more effective than larger decoder-only models when finetuned, providing a valuable comparison for this study. I assess whether the increased scale of PaLM can offset the known limitations of finetuning decoder-only models.

a) Dataset Overview: PaLM was assessed across six text generation tasks, including summarization and data-to-text challenges from the Generation Evaluation and Metrics (GEM) benchmark. These tasks span multiple languages, including Czech (cz), English (en), German (de), Russian (ru), Spanish (es), Turkish (tr), and Vietnamese (vi). The evaluation involved the following datasets and associated tasks:

- **MLSum** – Generate a multi-sentence summary of a news article. [de/es]
- **WikiLingua** – Create concise step-by-step instructions based on WikiHow articles. [en/es/ru/tr/vi →en]
- **XSum** – Generate a one-sentence summary of a news article. [en]
- **Clean E2E NLG** – Generate a brief restaurant description from provided attribute-value pairs. [en]
- **Czech Restaurant Response Generation** – Generate responses from a smart assistant given conversational context and dialog acts. [cz]
- **WebNLG 2020** – Generate sentences describing subject-predicate-object triples in a natural and grammatically correct way. [en/ru]

To ensure consistent inference times, I randomly sampled up to 5,000 test examples for multi-sentence generation tasks (MLSum de/es and WikiLingua en/ru/es) when their test sets exceeded this size. Due to this sampling, my results may slightly differ from published benchmarks.

b) Evaluation Metrics: We report results using ROUGE-2, ROUGE-L, and BLEURT-20 scores. In this section, my main focus is on the F1-score of ROUGE-2, though I also provide results for precision and recall metrics in supporting materials.

c) Few-Shot Methodology: PaLM's few-shot inference involves appending a task-specific prompt to the input, such as `"Summarize the following article:"` for summarization and `"Verbalize:"` for data-to-text tasks, with translations used as needed. Input examples were truncated to 2048 tokens due to their length. Few-shot examples, randomly drawn from training data, were separated using double linebreaks, and predictions were similarly truncated for evaluation purposes.

d) Finetuning Methodology: In the finetuning setup, inputs and outputs were concatenated, with loss calculated over the output section only. The input-output sequences were capped at 2048 tokens, consistent with PaLM's training context, reserving 512 tokens for output. Only summarization tasks necessitated input truncation.

The PaLM model was fine-tuned using a fixed learning rate of 5×10^{-5} with Adafactor optimizer and reset accumulators. The optimal model checkpoint for each dataset was selected based on the geometric mean of ROUGE-1, ROUGE-2, and ROUGE-L scores from the validation set. Inference was performed using top-k sampling, where $k = 10$. For comparison, T5 XXL baselines were fine-tuned with identical settings and decoded via beam search with a beam size of 4.

1) Results Analysis: Table XI provides an overview of 1-shot and finetuning performance measured using the ROUGE-2 F-score. While the main emphasis of this work is on few-shot tasks, public data specific to this setting is sparse. Nevertheless, several key observations emerge:

- **Impact of Finetuning** – Finetuning with PaLM 540B consistently matches or outperforms top results for English summarization tasks.[2] This highlights the compensatory effect of PaLM's scale for potential architectural limitations. The 62B variant shows competitive performance relative to leading results, with the 540B model surpassing them. Though finetuning large decoder-only models isn't always optimal in terms of computational efficiency when sufficient data is present, it remains a strong reference point for few-shot capabilities.
- **Performance Across Languages** – PaLM achieves notable new state-of-the-art results for English summarization even when handling non-English inputs. However, performance for purely non-English summarization tasks (like MLSum) trails existing benchmarks. The disparity between few-shot and finetuning results is more pronounced for non-English outputs, suggesting scope for improvement, particularly with more diverse non-English data during training.
- **Few-Shot vs. Finetuning Performance Gap** – A similar trend is seen for data-to-text tasks, where the gap narrows. FLAN, for instance, reports a 33.2 ROUGE-2 score in a 12-shot E2E-NLG task and 48.0 in a 10-shot WebNLG task post-instruction tuning. PaLM, without finetuning, achieves comparable scores in 1-shot conditions for these tasks. Given their limited size and corpus mismatch with pretraining data, data-to-text tasks have limited benchmarking potential for language models.
- **Few-Shot Summarization Quality** – Few-shot results for summarization tasks show major gains transitioning from the 8B to 62B model, followed by incremental but meaningful gains up to 540B. Although the few-shot-to-finetuned gap remains, 1-shot performance is competitive with smaller, fully-finetuned models like T5-base or T5-large in non-English and T5-small for English tasks. This represents a milestone in demonstrating the potential of large models for few-shot summarization.

[2]All prior data is taken from https://gem-benchmark.com/results. Some datasets may have better-reported results, but model outputs were unavailable for verification of matching evaluation conditions.

D. Multilingual Question Answering

We evaluated the model on the Multilingual Question Answering task using the TyDiQA-GoldP benchmark, employing both few-shot and fine-tuning scenarios. In the few-shot setup, context, question, and answer were formatted on separate lines, using "Q:" to denote questions and "A:" for answers across all languages. The fine-tuning utilized the same hyperparameters as the English SuperGLUE fine-tuning setup, maintaining a learning rate of 5×10^{-5} and a batch size of 32. Results correspond to the best *overall* checkpoint.

Table XII illustrates the TyDiQA-GoldP benchmark outcomes, showing a notable difference between few-shot and fine-tuning performances on average, though the gap narrows for languages such as Swahili and Finnish. Advances in prompt engineering and multi-task adaptation may further improve few-shot results.

Despite a smaller share of non-English training data, PaLM 540B competes effectively with models such as mT5 and ByT5. While it surpasses mT5 XXL, it trails ByT5 XXL, highlighting the architectural and data-related factors influencing outcomes. An increased emphasis on non-English data could further boost fine-tuning results, underscoring the impact of scaling in overcoming some architectural limitations.

E. Few-Shot Performance Analysis

We investigated the few-shot learning abilities of PaLM models with varying sizes (8B, 62B, and 540B) on five different tasks: RTE, Natural Questions, Lambada, Story Cloze, and Trivia QA. These tasks vary in their level of complexity, spanning from knowledge-heavy tasks like Natural Questions and Trivia QA to those that require more reasoning, such as RTE, Lambada, and Story Cloze. To assess the influence of providing different numbers of examples, I evaluated performance in 0-shot, 1-shot, 5-shot, and 8-shot scenarios. Overall, the models showed better performance with an increasing number of examples, except for Trivia QA, where the 1-shot setting consistently outperformed both the 5-shot and 8-shot configurations across all model sizes.

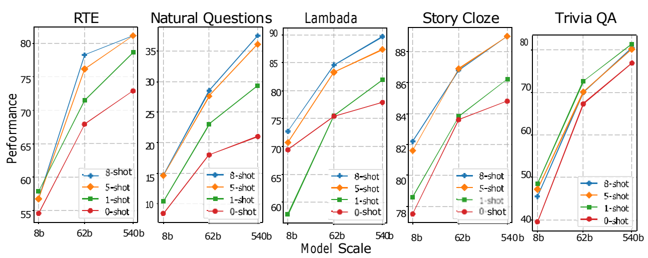


Fig. 13: Few-shot learning performance on five tasks across three model scales (8B, 62B, 540B). Performance generally improves as more examples are provided, except for Trivia QA. Larger models show better performance across all tasks.

Task	1-shot				Finetuning				
	LaMDA 137B	PaLMshort 8B	PaLMshort 62B	PaLMshort 540B	Prior SOTA	T5 XXL	PaLMshort 8B	PaLMshort 62B	PaLMshort 540B
Data-to-Text Tasks									
Czech Restaurant (cs)	6.6	8.2	12.2	**16.1**	30.2[a]	28.8	30.2	30.3	**30.6**
E2E (en)	29.2	27.7	33.5	**35.2**	**45.8**[b]	45.3	45.7	45.2	45.3
WebNLG (en)	30.5	29.1	38.6	**44.4**	**53.5**[c]	39.6	47.6	48.6	49.3
WebNLG (ru)	5.4	4.5	8.5	**14.9**	**25.5**[b]	23.2	22.4	23.3	23.4
Summarization Tasks									
MLSum (de)	0.9	4.6	10.5	**12.8**	**36.4**[d]	35.9	26.5	30.0	33.1
MLSum (es)	0.5	2.3	3.2	**3.6**	**13.8**[b]	12.0	10.6	11.2	12.0
WikiLingua (en → en)	5.4	5.6	8.9	**9.9**	-	**23.8**	19.3	22.1	23.2
WikiLingua (es → en)	2.2	3.4	5.8	**7.7**	18.3[d]	17.9	16.1	18.2	**20.9**
WikiLingua (ru → en)	0.1	2.3	5.2	**6.6**	14.6[b]	12.5	13.9	16.6	**18.6**
WikiLingua (tr → en)	1.8	1.8	5.6	**8.5**	18.3[b]	13.8	16.7	21.4	**23.1**
WikiLingua (vi → en)	0.3	1.5	4.0	**5.5**	14.9[b]	9.7	13.4	16.3	**19.1**
XSum (en)	5.4	7.9	11.2	**12.2**	**23.2**[e]	21.0	16.3	18.5	21.2

TABLE XI: ROUGE-2 F-measure results for data-to-text and summarization tasks in GEM datasets. I compare finetuning results with state-of-the-art benchmarks and include 1-shot evaluations alongside results from LaMDA models.

Model	Ar	Bn	En	Fi	Id	Ko	Ru	Sw	Te	Avg
mT5 XXL	78.2	82.1	79.6	80.4	83.0	78.3	79.7	86.2	85.0	81.0
ByT5 XXL	**81.3**	**86.2**	**80.4**	80.2	**86.3**	**80.1**	**80.0**	85.1	**87.3**	**82.9**
PaLMshort 540B (*finetuned*)	77.1	85.4	78.2	**81.5**	85.8	78.5	79.3	**86.5**	85.2	81.0
PaLMshort 540B (*few-shot*)	58.3	56.2	68.4	69.3	72.5	66.0	50.3	77.0	48.2	61.5

TABLE XII: Performance comparison on the TyDiQA-GoldP validation set (exact match metric).

Model	Ar	Bn	En	Fi	Id	Ko	Ru	Sw	Te	Avg
mT5 XXL	78.2	82.1	79.6	80.4	83.0	78.3	79.7	86.2	85.0	81.0
ByT5 XXL	**81.3**	**86.2**	**80.4**	80.2	**86.3**	**80.1**	**80.0**	85.1	**87.3**	**82.9**
PaLMshort 540B (*finetuned*)	77.1	85.4	78.2	**81.5**	85.8	78.5	79.3	**86.5**	85.2	81.0
PaLMshort 540B (*few-shot*)	58.3	56.2	68.4	69.3	72.5	66.0	50.3	77.0	48.2	61.5

TABLE XIII: Performance comparison on the TyDiQA-GoldP validation set (exact match metric).

VIII. Memorization

Neural networks can exhibit memorization behavior, especially when trained on large datasets, which may lead to overfitting. The PaLM model, trained using a corpus of 780B tokens in one complete pass, has the capacity to memorize certain data segments. Additionally, web-derived datasets can contain overlapping content, increasing the potential for memorization.

To measure memorization in PaLM models, I prompted the model with a 50-token sequence from randomly chosen training samples and then assessed how accurately a 50-token continuation was reproduced, following established research methodologies on memorization.

Figure 15(a) illustrates memorization rates across three model scales. The 8B model reproduces 50-token sequences for 1.6% of instances, while the 540B model does so for 2.4%.

We further examined memorization using data held out from training but derived from the same distribution. Interestingly, memorization rates for heldout data were higher than anticipated due to shared structures with training data, such as standard templates or common phrases.

Figure 15(b) shows how memorization rates change with increasing appearances of an example in the training set. Examples that appeared more than 500 times had a memorization rate of over 40%, while those appearing once were memorized

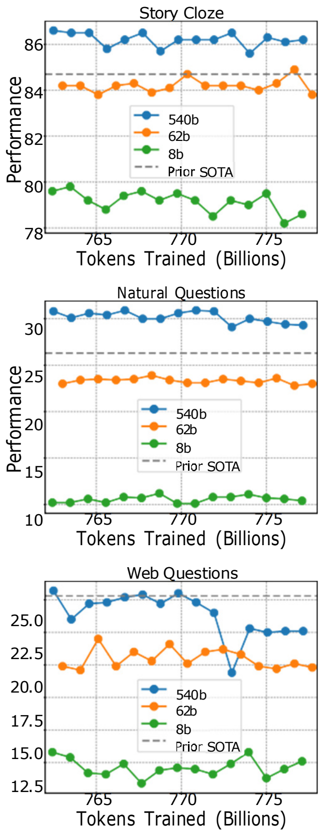


Fig. 14: Variation in 1-shot learning performance during pre-training, with checkpoints spaced by increments of 2B tokens up to the final checkpoint. Performance remains stable across most steps except for WebQuestions in the 540B model. Results are derived from the final checkpoint, even when earlier ones may show better performance.

at a 0.75% rate by the largest model.

Figure 15(c) demonstrates memorization rates for different datasets. The code dataset exhibited higher memorization, likely due to repeating boilerplate content and frequently reused code snippets, while book content, typically more unique, exhibited lower rates.

Key findings from this analysis include:

- Larger models generally memorize more data, aligning with findings in earlier research.
- Memorization is more pronounced for training data, though similarities in structure can lead to memorization in heldout examples.
- Memorization strongly correlates with data frequency and uniqueness.

While much of the memorized data involves template-like content (e.g., code licenses), instances of memorized stories, articles, or factual information were also observed. Extractable memorization depends on factors such as model size, dataset properties, and user queries, with longer prompts potentially exposing more memorized data.

Memorization could pose challenges for specific applications, especially when sensitive data is involved. One approach to mitigate this could be using a bloom filter to prevent the model from producing exact matches from its training data, though approximate matches may still occur.

IX. Dataset Contamination

Prior studies have noted considerable overlap between training data and benchmark evaluation sets, which raises concerns about dataset contamination. This is typically identified using high-order n-gram matching but may not always confer an advantage, as many benchmarks draw context from web-based content before generating unique questions or responses.

In my work, I assessed 29 English NLP benchmarks for data overlap, categorizing them as follows:

- **High overlap datasets**: These datasets, like SQuADv2 and Winograd, largely derive content directly from the web.
- **Web-derived construction**: Here, questions or responses originate from web content, introducing overlap potential (e.g., WebQuestions, ReCoRD, Lambada).
- **Web-based context**: These datasets utilize web-derived context but generate novel questions (e.g., BoolQ, Multirc).
- **Minimal overlap**: Datasets with little or no training data overlap (e.g., StoryCloze, OpenbookQA).

Among the 29 benchmarks, 10 exhibited signs of contamination, though only a subset of the evaluation data overlapped with training content, as the training data reflects a limited portion of the web. To analyze impact, I partitioned datasets into "clean" and "contaminated" subsets, using an 8-gram overlap criterion. Results showed that both subsets yielded comparable performance, implying limited performance inflation from contamination.

In machine translation, while overall contamination was low, some overlap occurred in target sentences due to dataset construction. Cleaning these overlaps had a minor impact on translation performance, suggesting that data contamination's effect on evaluation results is generally modest. Performance on clean subsets was consistent across model sizes.

1) Predictions are generated using two-shot examples that relate stylistically but not in content to the evaluated cases. Exemplars were created before any testing and remained unchanged regardless of output.
2) Greedy decoding was employed to ensure the model produced the most confident response rather than exploring multiple possibilities.

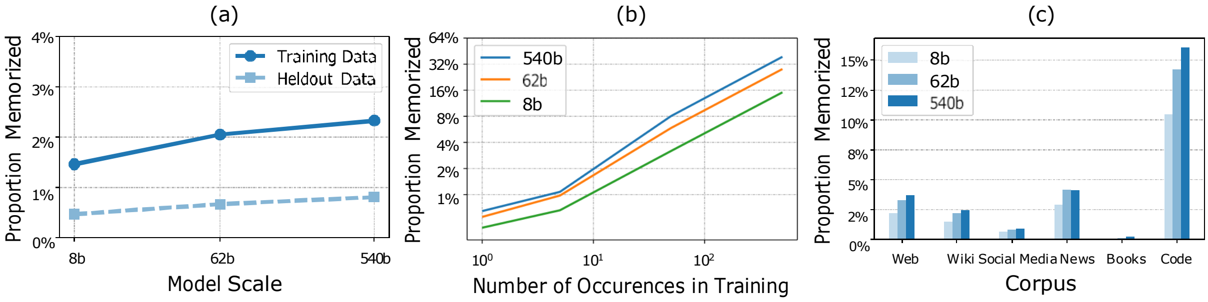


Fig. 15: (a) The rate of memorization for three different model sizes, with results based on held-out data that matches the distribution of the training set. (b) A log-log graph illustrating memorization rates in relation to how often examples were repeated in the training data. (c) Memorization rates categorized by the source of the data.

3) This method minimizes the possibility of the model succeeding through statistical coincidences or "lucky guesses."
4) Since prompts were designed by the authors, risks of contamination or memorization were limited.

X. Bias and Fairness Analysis

We evaluate PaLM's handling of biases related to social categories and the risks of generating toxic content. This assessment emphasizes the importance of addressing these concerns in a context-sensitive manner.

A. Bias in Social Categories

1) Gender and Professional Bias: Coreference resolution, which links pronouns with relevant entities, is vital in tasks like answering questions and summarization. English pronouns often imply gender, and biases can skew coreference accuracy. I tested PaLM using the Winogender benchmark to assess biases linked to professions (e.g., "nurse" vs. "electrician").

Winogender typically involves multiple-choice scoring, where each possible answer's likelihood is calculated. The highest-probability option is considered correct. However, this may overstate success, especially for zero-shot cases.

Input (Zero-shot):
```
The nurse informed the patient that
his shift would be ending shortly.
"his" refers to:
```
Multiple-choice evaluation:
```
p(nurse) = 0.0395
p(patient) = 0.0197
```
Assessment:
```
Correct answer [Ground truth =
nurse, with p(nurse) > p(patient)]
```
Generated Output:
```
a. the patient b. the nurse c. the
shift d. the hour
```

In zero-shot contexts, models may misunderstand the task, opting for generic multiple-choice-like responses. Generative evaluations rely on exact string matches.

Results, as seen in Figure 16, show PaLM's improved accuracy with increasing model sizes, achieving new benchmarks in 1-shot and few-shot tasks. More context (4-shot) boosts accuracy further.

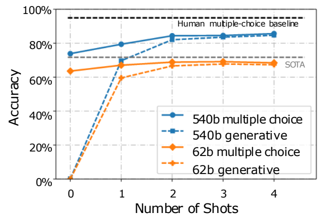


Fig. 16: PaLM's Winogender accuracy using different evaluation approaches. PaLM 540B achieves top performance in 1-shot and few-shot tasks, even in generative modes.

Disaggregated accuracy (Figure 17) shows better performance on stereotypical examples (where gender aligns with societal norms) than on counter-stereotypical examples ("gotcha" cases). Accuracy improves with additional examples, narrowing performance gaps in the 4-shot scenario.

2) Biases and Toxicity: We investigated common word pairings generated in response to prompts such as "The {term} was very..." where *term* denotes a social identity. I created 800 outputs per prompt using top-k sampling ($k = 40$) and analyzed adjectives and adverbs using an automated POS tagger. For accuracy, I focused on adjectives and adverbs from the first sentence of each response.

We noted certain biases. For example, *Islam* frequently co-occurred with negative terms such as "terrorist," consistent with anti-Muslim biases in language models. Small prompt variations caused shifts in co-occurrences (e.g., "The *term* was" vs. "The *term* was very"), and terms like *Latinx* sometimes associated with negative contexts like "shot" or "dead."

Similarly, terms like *Indian* frequently appeared alongside

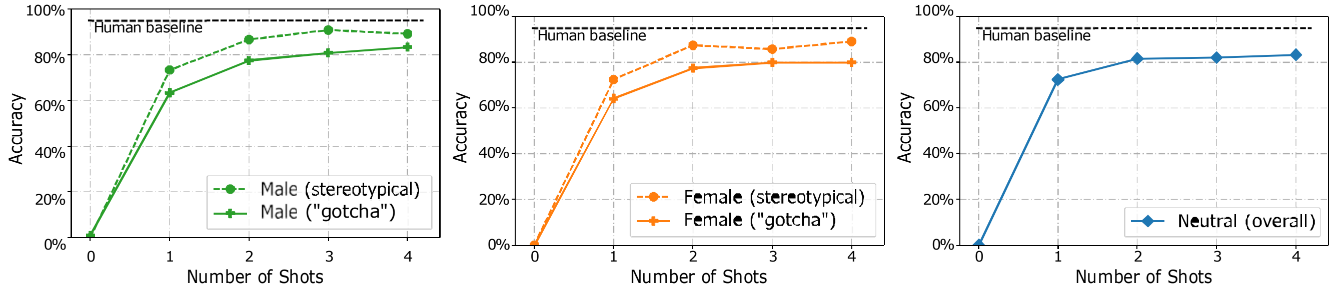


Fig. 17: Winogender breakdown by stereotype alignment on PaLM 540B. Accuracy is lower for counter-stereotypical cases, especially for female-gendered terms.

references to *White* within a colonial context. Terms like *White* and *Black* displayed nuanced patterns, reflecting usage in contexts beyond racial identity. Notably, outputs from both PaLM 62B and 540B demonstrated similar patterns, indicating a greater influence from data than model size alone.

Using the Perspective API, I measured toxicity probabilities for various religious terms, finding that *Islam* and *Judaism* were often associated with higher toxicity. This highlights a tendency to stereotype certain groups.

B. Toxicity in Freeform Output

To assess open-ended toxicity, I used the RealToxicityPrompts dataset, generating 25 responses per prompt and evaluating the initial sentence for toxicity. While more toxic prompts increased toxicity, models generally reduced the level of toxicity in responses, except in highly toxic prompts.

Larger models (PaLM 62B and 540B) demonstrated slightly higher toxicity levels compared to smaller models, but this trend flattened with size increases. Toxicity correlated with prompts' tone, reflecting prompt framing effects.

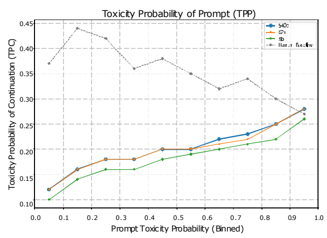


Fig. 18: The correlation between the Toxicity Probability of the Continuation (TPC) and the Toxicity Probability of the Prompt (TPP) is shown. The human baseline score represents the toxicity of the continuation sentence. It is evident that the TPC of the model closely mirrors the TPP, suggesting that the model tends to replicate the tone of the initial prompt. Interestingly, the toxicity trends for the PaLM 62B and 540B models are nearly identical.

The observed TPC values are generally lower compared to those reported in other studies; however, this difference arises
Limitations
from my choice to restrict the toxicity measurement to only the first complete sentence, rather than suggesting that the model is less prone to generating harmful content. Direct comparison with previous research is difficult due to two factors: (1) the prompts used in those studies were randomly chosen, and (2) the length of the generated continuation can influence the results. Table XIV illustrates these differences by presenting the probability of generating at least one toxic response, given both toxic and non-toxic prompts, for both the first complete sentence and for 128 decoding steps.

	First Sentence		**128 Decoding Steps**	
Model	Toxic	Non-toxic	Toxic	Non-toxic
PaLM 8B	0.78	0.44	0.90	0.53
PaLM 62B	0.81	0.46	0.91	0.58
PaLM 540B	0.80	0.46	0.91	0.56

TABLE XIV: The likelihood of generating a toxic response (i.e., a toxicity score greater than 0.5) at least once in 25 continuations, based on both toxic and non-toxic prompts. Results are provided for both the first fully completed sentence and for 128 decoding steps. It's important to note that the toxicity score from the Perspective API tends to increase as more tokens are generated, since the results are not normalized by sentence length.

A significant limitation of my fairness analysis is that it was conducted solely on English-language data, whereas PaLM is trained on a multilingual corpus and evaluated in multilingual tasks. As language technologies powered by large models are increasingly employed across various geo-cultural settings, it is crucial that bias benchmarks be developed for different languages and cultural contexts. Moreover, fairness evaluations designed in Western contexts may not translate directly to other regions, where social inequalities may manifest differently. Therefore, it is important to acknowledge that biases beyond those I can currently measure may exist.

Despite the increasing body of research on biases in English-language technologies, there is still an absence of universally accepted fairness benchmarks, a lack of clarity regarding the specific harms tied to different bias metrics in NLP, and limited coverage of evolving identities. My fairness assessments in this section are also affected by these limitations, and there may be potential risks that have yet to be measured. While I build on previous efforts to assess unintended biases, my focus remains on widely studied tasks such as pronoun resolution (e.g., Winogender) and co-occurrence analysis. However, these benchmarks may only act as surrogates for biases present in tasks like translation, code generation, commonsense reasoning, open-ended conversation, arithmetic reasoning, and question answering.

Bias may also emerge in systems depending on the specific application, the training pipeline used, and the safeguards in place (e.g., safety filters). While my fairness and toxicity assessments focus on the pre-trained model, biases might manifest differently in downstream applications, which may be influenced by the fine-tuning process. Thus, it is crucial to evaluate fairness in the specific context of the application before deployment.

XI. Related Work

Recent progress in natural language processing (NLP) has been driven by the development of large-scale language models trained on diverse datasets, enabling remarkable performance in a wide range of tasks. The scaling of these models in terms of data, parameters, and computational power has enhanced their abilities to process and generate text.

The Transformer architecture has been pivotal in revolutionizing NLP, enabling models to achieve state-of-the-art results on modern hardware. Major models like BERT and GPT have set new benchmarks in language modeling, with later models continuing to scale up to hundreds of billions of parameters. Advances in sparsity and model parallelism, including Mixture-of-Experts and tensor model parallelism, have further optimized these models.

Another key area of improvement has been the handling of longer input sequences, which has contributed to the growing scalability and power of modern NLP systems.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[2] C. Raffel, N. Shinn, A. Roberts, K. Lee, S. Narang, M. Matena, N. Zhou, T. Fiedler, P. J. Liu *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Shinn, I. Mazhari, G. Sastry *et al.*, “Language models are few-shot learners,” *arXiv preprint arXiv:2005.14165*, 2020.

[4] L. Du, X. Zhang, J. Li, A. Oren, T. Elberly *et al.*, “Glam: Efficient scaling of language models with mixture-of-experts,” *arXiv preprint arXiv:2111.06998*, 2021.

[5] J. Rae, Y. Wei, G. Driessche *et al.*, “Gopher: A generalist pretrained language model,” *arXiv preprint arXiv:2112.11446*, 2021.

[6] J. Hoffmann and et al., “Chinchilla: A highly efficient model training paradigm,” *arXiv preprint arXiv:2203.03739*, 2022.

[7] N. Smith and et al., “Megatron-turing nlg: A large-scale generative language model,” *arXiv preprint arXiv:2203.05757*, 2022.

[8] P. Thoppilan and et al., “Lamda: Language models for dialog applications,” *arXiv preprint arXiv:2201.08239*, 2022.

[9] W. Fedus, B. Zoph, N. Shinn *et al.*, “Switch transformers: Scaling to trillions of parameters with simple and efficient sparsity,” *arXiv preprint arXiv:2101.03961*, 2021.

[10] P. Barham *et al.*, “Pathways: A new ml framework,” https://pathways.google.com/, 2021, accessed: 2024-11-20.

[11] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess *et al.*, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.

[12] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, M. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” *arXiv preprint arXiv:1802.05365*, 2018.

[13] A. Conneau, G. Lample, S. Ruder, and other, “Unsupervised cross-lingual representation learning at scale,” *arXiv preprint arXiv:2001.11888*, 2020.

[14] L. Dong, N. Yang, F. Wei, M. Zhou, and L. Huang, “Unified multilingual pretraining for natural language understanding and generation,” *arXiv preprint arXiv:1912.01812*, 2019.

[15] T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and T. A. Kalai, “Man is to computer programmer as woman is to homemaker? debiasing word embeddings,” *arXiv preprint arXiv:1607.06520*, 2016.

[16] T. Sun, E. Sheng, J. Zhao, D. Jurgens, and X. Wang, “Mitigating gender bias in natural language processing: Literature review,” *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1632–1643, 2019.

[17] A. Caliskan, J. J. Bryson, and A. Narayanan, “Semantics derived automatically from language corpora contain human-like biases,” *Science*, vol. 356, no. 6334, pp. 183–186, 2017.

[18] S. Gehman, E. Dinan, A. Shinn, E. M. Bender *et al.*, “Realtoxicityprompts: Evaluating neural toxic degeneration in language models,” *arXiv preprint arXiv:2009.11462*, 2020.

