DSA412: Character-Level Large Language Models

November 21, 2025

Group Composition:

1. Tan Kai Hong Matthias, A0252489R, e0958077@u.nus.edu
2. Robbie Chia Yuan Ee, A0252778N, e0958366@u.nus.edu

1 Introduction

Large language models have grown rapidly in scale, but in this project we aim to build a small, character-level transformer model under strict real-world constraints: limited GPU compute, time, and data. The task is next-character prediction on the Text8 dataset, which allows us to study modelling decisions in a controlled setting. Rather than competing on scale, we focus on understanding how each architectural and training component contributes to model performance. The final model we train is the result of a series of experiments designed to balance performance with compute efficiency.

1.1 Problem Statement

Given a text corpus $D = (x_1, x_2, \ldots, x_T)$, where each x_t represents a character from the text8 dataset, our objective is to train a model f_θ that learns the conditional probability distribution:

$$P_\theta(x_{t+1} \mid x_{t-L+1}, \ldots, x_t),$$

for a context window of length L. It requires the model to learn the structure of text at a granular level and predict the most likely next character based on the preceding L characters.

1.2 Dataset

We use the **Text8** dataset, a 100M-character cleaned WIikipedia dump containing only lowercase letters and spaces. We follow the standard 90/10 split - 90M characters for training/validation and a 10M held-out test text. To preserve text structure, we made sure to make splits only at whitespace boundaries.

1.3 Goals of the Project

The goals of our project are

- Implement an efficient transformer-based character model and maximize its accuracy
- Design an experiment framework that works under strict compute limits
- Determine an optimal sequence length L and architecture through systematic ablations and tuning

2 Transformer Model Architecture

Since our task is purely auto-regressive text generation, we use a **decoder-only Transformer** [VSP+17], omitting the encoder entirely as the model only needs to predict the next character from previous ones. The architecture, shown in Figure 1, consists of token embeddings that map characters into d_{model} dimensional vectors, positional encodings to provide order information,

and a stack of n_{layers}. Each block contains causal self-attention where tokens attend only to earlier positions implemented using multi-head attention,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

followed by a position-wise feed-forward network with GELU activation. Layer normalization stabilizes training, and the final projection maps hidden states to logits over the 27 character vocabulary. This architecture is sufficient for efficient next-character prediction on text8.

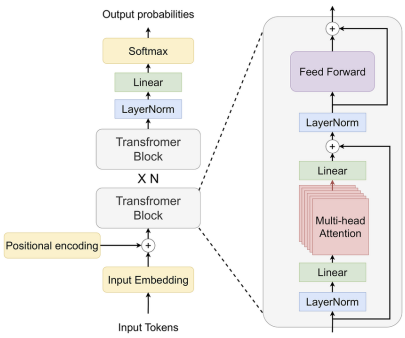

Figure 1: Decoder-only Transformer

3 Experiment Framework

3.1 Justifications

Our experiments were constrained by free cloud GPUs (e.g Colab T4, Kaggle), which impose runtime limits and occasional interruptions. To manage this, we set a compute budget of 1-1.5 hours per experiment and implemented a fail-safe checkpoint system that saves weights and optimizer states periodically. Additionally, we built a function *calculate_throughput()* to determine the maximum number of training steps possible within the allocated budget. This allowed us to design experiments that were both reproducible and computationally feasible.

3.2 Baseline Model

Some preliminary tests showed that a ≈ 3.2M parameter model fitted well within our compute budget. We deliberately avoid larger architectures of 100M+ parameters like GPT2-Small (124M) or SmolLM2 (125M), because training them meaningfully would require billions of tokens, far more than the 100M available in Text8. A smaller model ensures manageable training times, reduces overfitting risks, and provides a stable foundation for controlled ablation studies.

4 Experiment Results

Starting from our 3.2M–parameter baseline, we conducted a structured sequence of ablation experiments to quantify the effect of individual architectural and training decisions. This was followed by targeted hyperparameter tuning to refine the model under our compute constraints.

4.1 Ablations

All comparisons used **validation loss** as the primary metric. Since our model is trained with cross-entropy, validation loss directly reflects the negative log-likelihood objective and provides

the most sensitive measure of model quality. Given hardware limits, we use a **one-factor-at-a-time (OFAT)** strategy where in each experiment we vary a single component while holding all others fixed. This mirrors the approach adopted in scaling studies such as Chinchilla [HBM+22], Gopher [RBC+21], and PaLM [CND+22], where isolating factors is necessary due to high computational cost. After each experiment, we select the best-performing configuration as the new default for subsequent tests. Thus, our experiments follow a staged design, where each category informs the next:

1. **Vary foundational Architectural Components** — loss function settings, auxiliary losses, optimizer, positional encoding, attention type.
2. **Model Scaling** — $d_{\text{model}}, n_{\text{layers}}, n_{\text{heads}}$.
3. **Context Length** - L.

4.1.1 Vary foundational Architectural Components

Across our architectural ablations, most variants converged to similar validation losses (≈ 1.3–1.5), but several components consistently performed better than others as seen in Fig 2 and 3.

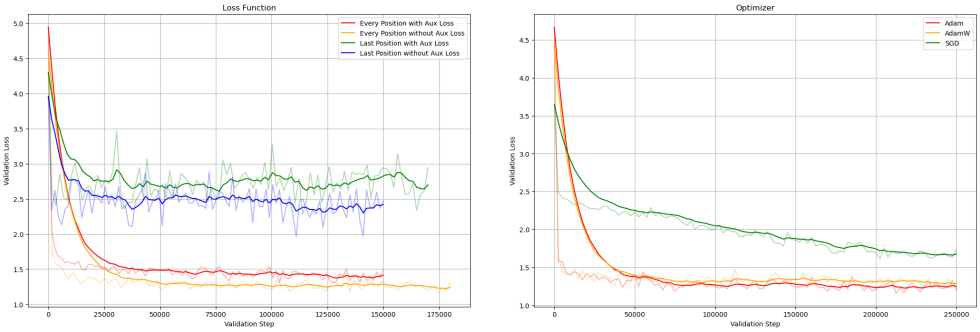

Figure 2: Validation loss curves for Loss Function (left) and Optimizer (right).

For the **loss function**, computing cross-entropy at every position produces smoother and lower validation loss than last-token-only loss, while auxiliary losses introduced instability. For the **optimizer**, Adam and AdamW (with weight_decay = 0.1) clearly outperformed SGD. Since Adam matched AdamW's performance with slightly smoother behaviour and fewer regularisation complications, we selected Adam as our default, leaving weight decay to hyperparameter tuning.

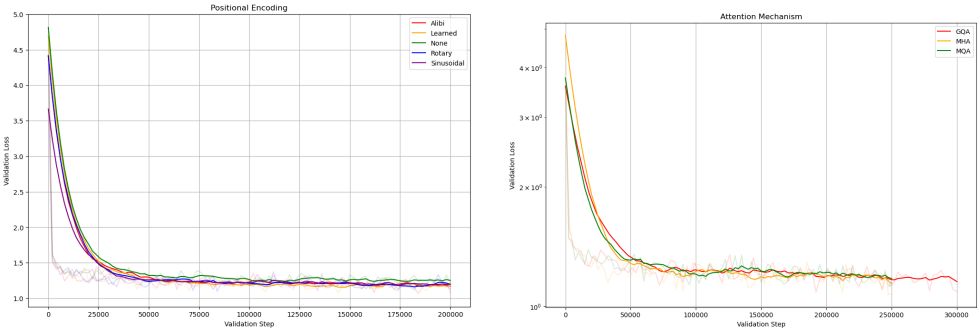

Figure 3: Validation loss curves for Positional Encoding (left) and Attention Mechanism (right).

The **positional encoding** ablations showed only minor differences across sinusoidal, learned, rotary, and ALiBi embeddings. Rotary performed marginally more smoothly overall and was chosen as a reliable, low-risk option for our small model. Similarly, for **attention mechanism**, MHA, MQA, and GQA produced nearly identical curves, but GQA offered comparable performance with better parameter and compute efficiency, fitting our resource constraints. Overall, this led us to adopt a stable baseline configuration: cross-entropy (no auxiliary loss), Adam optimizer, rotary positional encoding, and GQA attention.

4.1.2 Model Scaling

After making the changes to our model architecture, we varied n_{heads}, d_{model}, and n_{layer} to understand how scaling affects performance under our fixed compute budget. Overall, scaling the model generally improved convergence, but the gains diminished quickly after a certain point, and larger configurations became significantly more costly to train as seen in Fig 4.

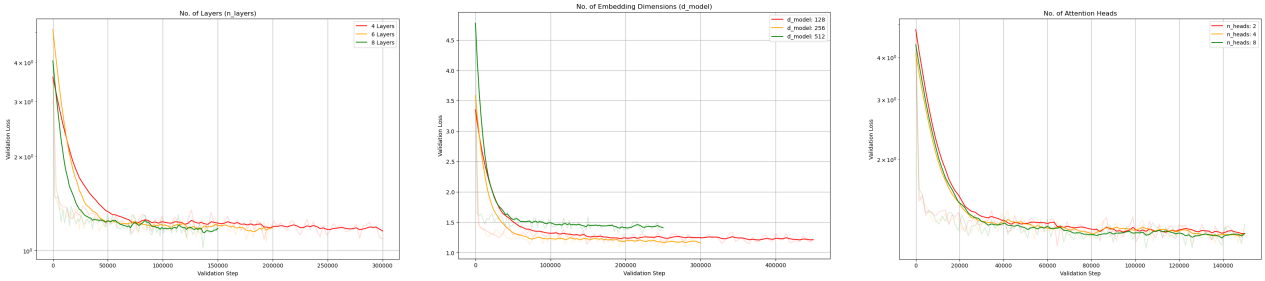

Figure 4: n_{layers} (left), d_{model} (middle), n_{heads} (right).

For n_{layers}, 6 layers improved upon 4, whereas 8 layers provided only marginal gains but required significantly more compute. Varying d_{model} had a clearer effect, 256 consistently outperformed 128, while 512 converged more slowly and plateaued higher due to reduced batch throughput. For n_{heads}, the models with 2, 4, and 8 heads showed nearly identical validation losses, indicating that additional heads offer little benefit at our small model scale. Given our resource constraints, the configuration $d_{\text{model}} = 256$, $n_{\text{layers}} = 6$, and $n_{\text{heads}} = 4$ offered the best trade-off between performance and efficiency.

4.1.3 Context Length L

We also wanted to optimize the context window L which corresponds directly with this model's sequence length. We trained the model with sequence length 64, 128 and 256 where the results can be seen in Fig 5.

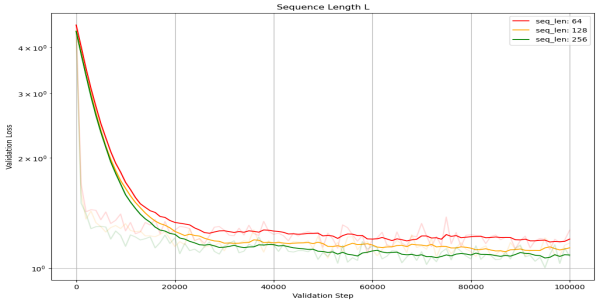

Figure 5: Validation loss Curves for Seq Len.

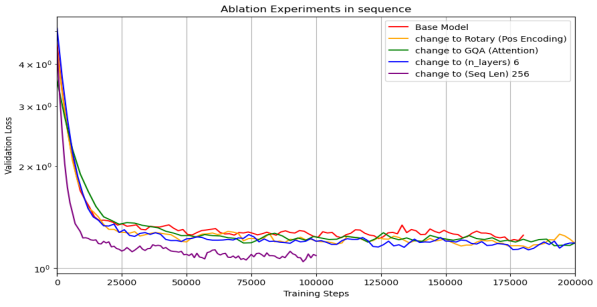

Figure 6: Ablation experiments in sequence.

From Figure 5, longer sequence lengths produces smoother and lower validation loss. This is expected, since increasing L allows the model to capture longer-range dependencies, even though it slows convergence and increases computational cost. Unlike scaling d_{model}, n_{layers}, or n_{heads}, which produced diminishing returns relative to compute, increasing L resulted in a clear improvement. Although 256 is more expensive, the performance gain justifies the trade-off, and we therefore selected $L = 256$ as the optimal context length. Putting all the changes together in 6, we see that the final architecture produces a direct performance gain (purple line).

4.2 Hyperparameter Tuning Results

The results for batch size, gradient clipping, dropout, and label smoothing are presented in Figure 7 and 8. These parameters were tested to balance model performance and training efficiency.

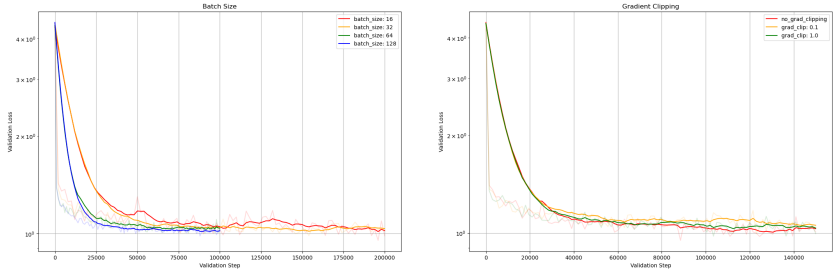

Figure 7: Batch size (left) and Gradient clipping (right).

Testing batch sizes from 16 to 128, we found that 32 provided the best balance between convergence speed and memory usage, as larger batches offered little improvement and smaller ones slowed training. For gradient clipping, we compared no clipping, 0.1, and 1.0, and observed that no clipping produced smoother and faster convergence, with clipped variants giving no benefit.

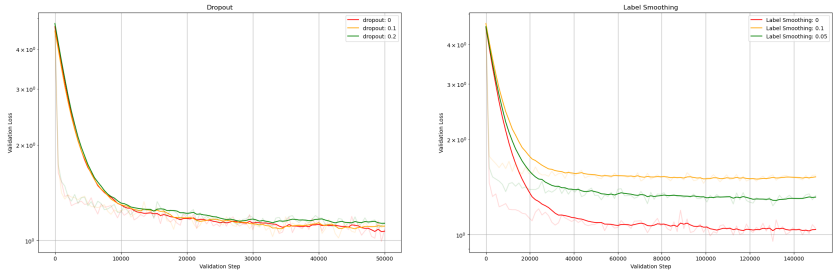

Figure 8: Dropout (left) and Label smoothing (right).

Dropout values of 0, 0.1, and 0.2 were tested, and 0 consistently achieved the lowest validation loss, likely because our model is small and not prone to overfitting. Similarly, label smoothing values of 0, 0.05, and 0.1 were evaluated, with 0 performing best. Thus, our final choices were: batch size 32, no gradient clipping, dropout 0, and label smoothing 0.

4.2.1 Bayesian Optimization

We then adopted Bayesian Optimization (BO) for the remaining hyperparameter search as our compute budget prevents exhaustive sweeps over the learning rate and weight decay space that grid or random search would require. We used **Optuna**, which is designed for limited compute

environments [ASY+19] and balances exploration and exploitation through pruning. Its optimization strategy is based on the Tree-Structure Parzen Estimator which models good and bad hyperparameters regions with separate density estimators and proposes new points where the ratio of these densities is most favourable. This scales well to categorical and conditional hyperparameters and is robust to the noisy objective landscape typical in deep-learning training.

Our search space included: (i) learning rate (LR) schedule $\in$ cosine, warmup decay, constant, (ii) LR $\in [0.0001, 0.003]$, (iii) weight_decay $\in 0.0, 0.01, 0.05, 0.1$, and (iv) warmup_ratio $\in [0.02, 0.2]$. The best trial achieved a validation loss of 0.972 with a warmup–decay schedule, learning rate $\approx 9.63 \times 10^{-4}$, weight_decay = 0.05, and warmup_ratio ≈ 0.131. Figures 9 and 10 summarise the experimental search behaviour.

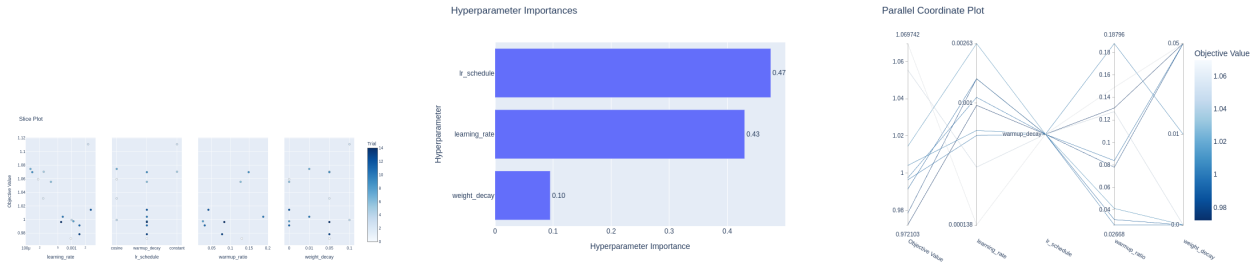

Figure 9: Slice plot, Hyperparameter importance plot, and Parallel coordinate plot from the Bayesian Optimization study.

The slice plot provides a direct view of sensitive regions: good trials cluster at LR ≈ 0.001–0.002, warmup–decay consistently yields lower losses than cosine or constant schedules, warmup ratios around 0.03–0.07 perform best, while weight decay shows no strong monotonic trend. Optuna's functional ANOVA (middle) quantifies these effects: LR schedule (47%) and LR (43%) dominate the variance in validation loss, confirming them as primary optimisation levers; weight decay contributes more modestly (10%). The parallel coordinate plot (right) illustrates joint structure in the best runs—low-loss configurations consistently use warmup–decay, LR near 10^{-3}, warmup ratio in the range 0.03–0.07, and weight decay of either 0.0 or 0.05.

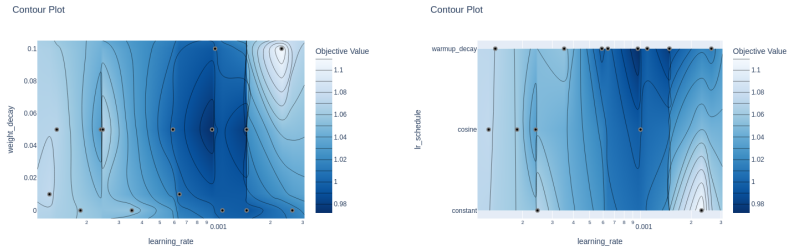

Figure 10: (Left) Learning rate vs. weight decay, (Right) Learning rate vs. schedule

The contour plots highlight key interactions. Warmup–decay exhibits a broad basin of low loss centred at LR ≈ 0.001, whereas cosine has a narrower optimum and the constant schedule performs consistently worse. The LR–weight decay landscape shows a low-loss region at LR ≈ 0.001 with weight decay between 0.01 and 0.05; both under-regularisation and over-regularisation lead

to degradation. These patterns support our choice of the best-found configuration for training the final model.

5 Discussion

5.1 Scaling Laws and Model Size

Midway through the project, we realised our models converged early across almost all ablations, often within a small fraction of the planned iteration budget. This behaviour aligns with transformer scaling-laws literature: when model capacity is small relative to dataset size, additional training steps provide diminishing returns [KMH+20, HBM+22]. On hindsight, our early experiments did not fully capitalise on this principle, and we had allocated generous iteration budgets while keeping model size fixed, wasting compute. We could have shorted early runs and redirected more compute towards scaling model width, possibly yielding stronger final performance while respecting resource limits.

5.2 Compute Budget and Throughput

We also had a mixture of good and bad compute budget planning. We initially relied on Colab's free T4 GPU, where our throughput-estimation function helped structure time budgeting accurately. As training was significantly slower than anticipated most times, we were able to obtain Colab Pro's higher tier resources. Among available accelerators, TPU v5e provided the best throughput-per-credit ratio as its systolic-array architecture delivered strong attention/MLP performance at far lower credit consumption than the A100, allowing us to run ablations, BO trials and the final training with a much larger step size. However, our throughput function consistently underestimated achievable iteration speeds on TPU v5e, leading to inaccurate planning and more compute could have been allocated to other parts. In retrospect, combining TPU v5e with conservative iteration limits from the beginning would have substantially improved overall efficiency.

5.3 Bayesian Optimization

We felt our BO pipeline performed well in discovering a consistent and interpretable optimum across multiple visualisations, with the TPE sampler handling conditional hyperparameters effectively. However, this came with setbacks as we experienced several failed BO runs due to configuration mistakes - including an early oversight where Adam was used instead of AdamW, causing weight decay to have no effect despite being treated as a tunable parameter. This led to misleading importance estimates before we corrected the setup. These iterations highlighted how sensitive BO studies are to seemingly minor implementation errors, and reinforced the importance of verifying each component before launching multi-hour runs.

5.4 Future Considerations

The most costly challenges came from configuration errors and insufficient incremental testing. Given the long runtimes, even with TPU v5e, every misconfigured trial wasted both compute and time. Future workflows would benefit from (i) stricter configuration validation before full runs, (ii) more aggressive use of small-scale "sanity-check" runs, (iii) periodic re-benchmarking of throughput after switching hardware and (iv) better integration of checkpointing/pruning mechanisms to avoid unnecessary computation.

Additionally, although we used BO only for the optimizer hyperparameters, extending BO to model-architecture ablations could have been computationally feasible if paired with early-stopping

or more granular throughput-aware training schedules. This is an avenue worth exploring in future work.

6 Final Model

After selecting the optimal hyperparameters from our BO study, we trained our final model over 1M iterations. The training and validation loss curves (Fig. 11) show stable convergence.

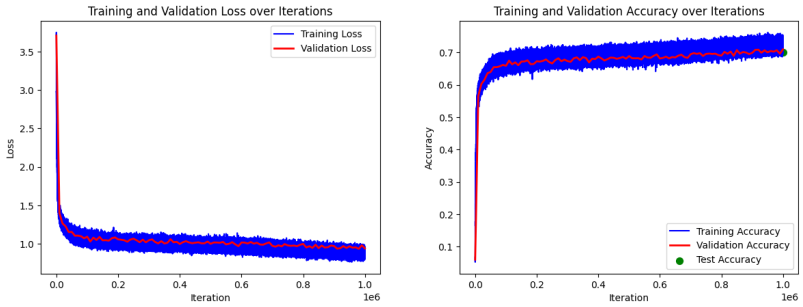

Figure 11: (Left) Final Loss Curve, (Right) Final Accuracy Curves with Test Accuracy

The training loss decreases smoothly throughout optimization, while the validation loss drops early and then plateaus, remaining stable for the remainder of training. This behaviour does not exhibit the widening gap characteristic of overfitting. Instead, it reflects typical deep-network optimization where the model continues to refine its fit on the training set while validatio loss stabiliozes once the representational capacity of the model is saturated. The narrow train-val gap indicates that extending training would offer only marginal improvements.

On the held-out test set (averaged across 100 batches to reduce sampling noise), the model achieved mean accuracy $= 70.07\%$, mean last-char accuracy $= 71.09$ and standard deviation of $= 0.87\%$. The low variance indicates stable generalization, and the test accuracy aligns closely with the validation accuracy curve in Fig. 11, confirming no overfitting. The slightly higher last-character accuracy reflects the benefit of full-context prediction in autoregressive models.

Overall, these results validate the architectural choices from ablations and the hyperparameters selected through Bayesian Optimization. The final model achieves $\approx 10\%$ accuracy improvement over our baseline, demonstrating successful optimization under strict compute constraints.

References

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *arXiv preprint arXiv:1907.10902*, 2019.

[CND+22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[HBM+22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl,

Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[KMH+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Ilya Sutskever, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[RBC+21] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sally Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lukasz Kaiser Jones, Ashish Chopra, M. Nuri, W. Zeng, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2017.