Forecasting AI Capabilities and Major Bottlenecks

Jay Chooi jeqin_chooi@college.harvard.edu
Ivan ("Dewey") To ito@college.harvard.edu
Henry Huang hhuang@college.harvard.edu
Christina Hu christinahu@college.harvard.edu

I. Introduction and Motivation

Recent advances in artificial intelligence and machine learning, particularly in large language models (LLMs), have been driven by many factors. In this work, our objective is to investigate the relationships between these factors and their impact on the performance of LLMs. Specifically, we seek to understand how algorithmic advancements, the computational resources allocated for training, and the size of the training datasets influence model performance on standardized benchmarks.

To explore these dynamics, we have compiled data from prominent LLMs released in recent years. This data includes details on the compute resources used, the size of the training datasets, and the models' performance on various benchmark tasks. We used Hugging Face's OpenLLM dataset, which provides a valuable source of data on the performance of AI models in real-world applications. By combining this with data from EpochAI, we can further investigate how these factors influence performance and identify any trends in how computation usage and data set size have evolved over time. Since we combined two datasets, we **explicitly state our research question** as: *given the state-of-the-art of model predictors, what is the state-of-the-art model performance*?

Our hypothesis is that each of these factors, including training and test computation, dataset size, model size, and algorithmic improvements (as estimated by date of publication), will have a positive correlation with model performance. Since this is a complex system, we aim to perform EDA and test the assumptions of ELIHN, attempt multiple transformations, use model selection techniques, such as cross validation, to choose a suitable model, and predict future LLM performance.

II. Data and EDA

We initially only used the EpochAI dataset, which can be accessed at EpochAI Notable AI Models. After loading our dataset and examining which training sets have been used most often, we found that WikiText-103 was the most commonly used language dataset, as we wanted an apples-to-apples comparison. Therefore, we restricted our analysis to the 21 models that use WikiText-103 as their training set. With this in mind, we set our performance metric to be a model's perplexity score on a testing subset of WikiText-103. (Perplexity is commonly used to measure a model's ability to predict the next word in a sequence. Lower perplexity scores are better than higher ones.) Unfortunately, EpochAI's dataset does not record the test perplexity scores of our language models, so we systematically reviewed 21 research papers to extract the test perplexity scores of the models. We have organized the data into a spreadsheet.

After conducting our initial EDA and subsequently meeting with our assigned TF, Gabe, we received feedback that ideally, the number of data points should be 10x the number of predictors. Thus, we found methods to expand our dataset.

We first explored the possibility of increasing the number of data points by expanding our dataset to include more AI models and not just those trained on WikiText. We then examined these models' performance on the GPQA benchmark from this dataset so that a standardized testing loss can be used as a metric for comparison. We also noticed that there are two datasets published for these models, Notable AI Models and Large-Scale AI Models. We combined both datasets into a single dataset.

After completing the dataset combination steps, we arrived at 40 models, which is almost a doubling from the 21 models we found in our initial EDA. However, if we consider the other predictors, including training compute and dataset size, we only obtained 16 models, which is unsatisfactory.

To solve the problem of an insufficient number of data points, we first identified the cause of the lack of data. Many models on the GPQA benchmark don't report their training dataset size or the amount of compute they were trained on. However, we do have data on many models' training compute, and data on many models' dataset size; the only problem is that the intersection set of models with both these data and the GPQA data is small.

We solved this problem by grouping models according to time windows. Specifically, we considered the top-N models in each aspect from training compute to test-time performance for every week from March 2022 to November 2024 due to data availability, setting up our data to specifically address our research question stated in the introduction. This allowed us to tap into the bulk of the data points provided by Epoch AI and utilize the Open LLM Leaderboard, provided by Hugging Face, to track test-time performance without requiring it to be done on specific models that we have training compute

and dataset size information on. After completing the time window subset step, we arrived at exactly ***139*** data points, coincidentally.

We started performing our secondary EDA, visualizing each predictor across the weeks since March 2022, shown in the Appendix. Since we are combining the two datasets, we wanted both to align in the number of parameters across the weeks since March 2022 as closely as possible. The initial distribution is shown in Fig. 1:

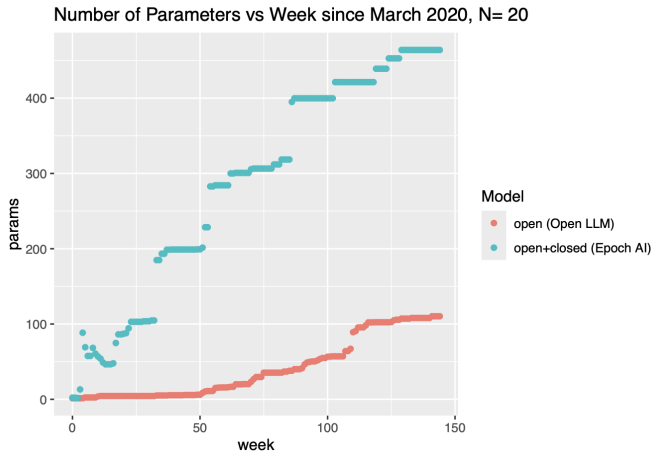


Fig. 1: Initial Parameters vs Weeks Graph

We noted that there is a gap between closed models and open models. To make it a fair comparison, we restricted the dataset to those with open weights or with API access so that it is feasible to be included in the Open LLM. This is visualized in Fig. 2.

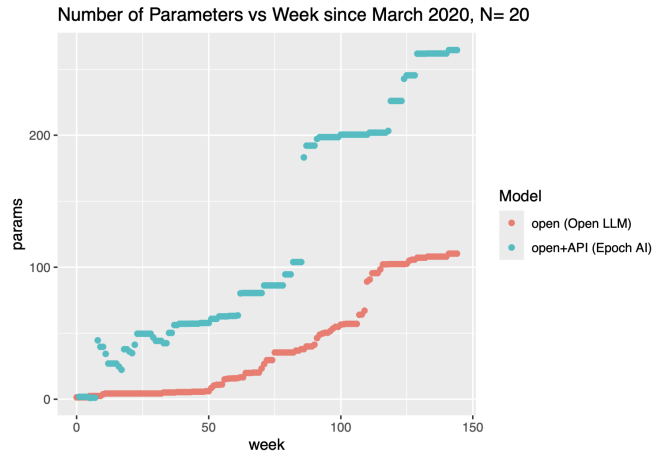


Fig. 2: Improved Parameters vs Weeks Graph

We saw that the gap is now visibly smaller. We further limited our models to open weights only, visualized in Fig. 3.

Note that the y-axis is now limited, with the models in the Epoch AI dataset reaching a maximum below 400. This

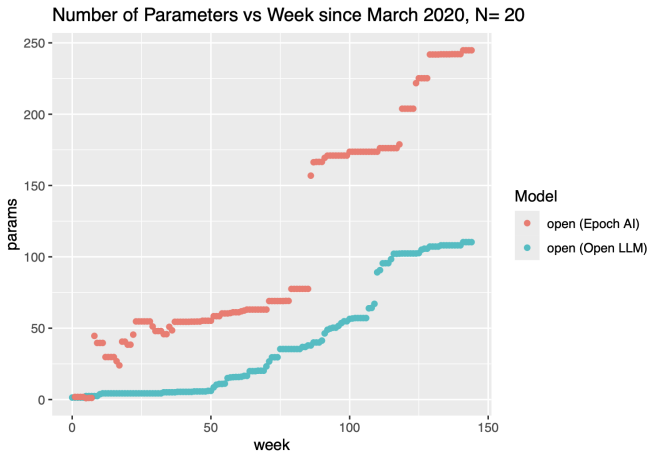


Fig. 3: Final Parameters vs Weeks Graph

motivated us to use the models with open weights only from Epoch AI. To make sure our data is free from NA, we subsetted from week 6 onwards.

Next, we began to build a model to predict performance on the Measuring Massive Multitask Language Understanding (MMLU) benchmark, visualizing the residual vs fitted values and QQ Residual plots, to evaluate our ELIHN assumptions. We used the base linear model, treating each predictor as in our initial exploratory analysis. Specifically, we fit the following model using the `lm` function in R:

```
baseline_model <- lm(formula = mmlus ~
    params + co2s + week +
    dataset_sizes + train_computes,
    data=dataset)
```

We obtained a baseline model summary of:

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.2908 -1.3384  0.0847  1.0528  4.1472

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -4.090e+00  5.160e-01  -7.926 8.06e-13 ***
params          3.360e-01  7.801e-02   4.307 3.20e-05 ***
co2s           -2.079e-01  1.232e-01  -1.688   0.0938 .
week            1.386e-01  1.342e-02  10.325  < 2e-16 ***
dataset_sizes   7.469e-12  9.353e-13   7.986 5.80e-13 ***
train_computes -3.327e-24  4.362e-25  -7.628 4.08e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.941 on 133 degrees of freedom
Multiple R-squared:  0.9905,    Adjusted R-squared:  0.9902
F-statistic:  2778 on 5 and 133 DF,  p-value: < 2.2e-16
```

In addition, we visualize the diagnostic plots in Figures 4, 5, 6. We note that the QQ-plot of this baseline is showcasing very slight signs of positive skew, as the standardized quantiles

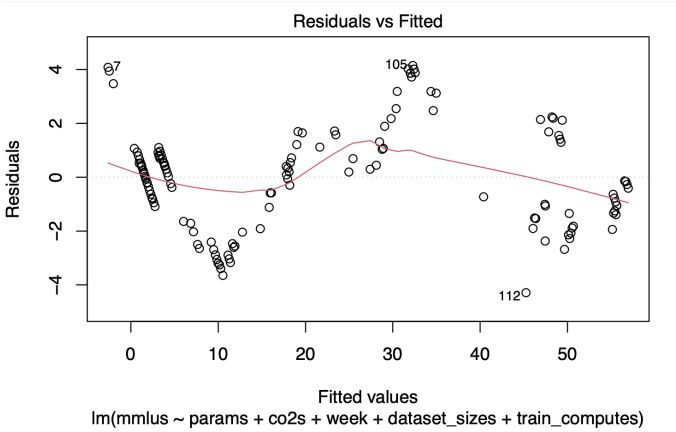


Fig. 4: Residuals vs Fitted Plot

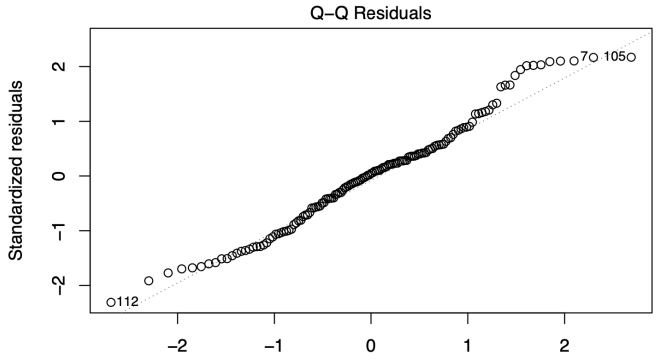


Fig. 5: QQ Plot

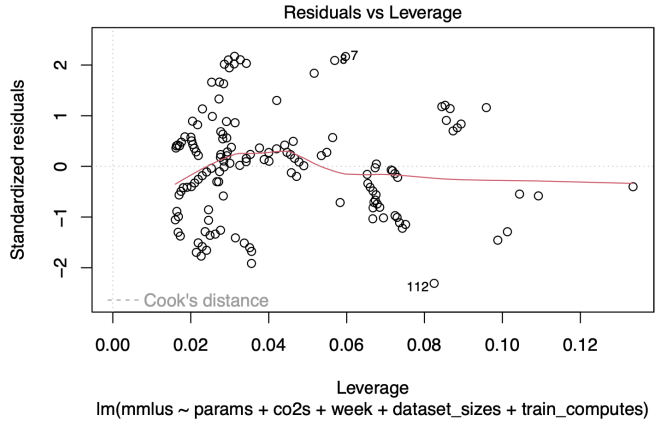


Fig. 6: Residuals vs Leverage Plot

on both the higher and lower end of our model's residuals are slightly higher than that of a standard normal, while the fitted vs residuals plot exhibits some potential patterns (e.g. the curve somewhat resembles a third order polynomial), so we can experiment with various transformations later on. Finally, we see that from the residuals vs leverage plot, the point labeled "112" appears to be a notable outlier, with both high leverage and a large magnitude residual. This suggests that this point may be highly influential in determining the model's fit.

The basic model including all predictors without transformations achieved an excellent fit with an adjusted R-squared of 0.9902. Note however that there are non-intuitive estimates, including negative coefficients for amount of CO2 emission and training compute. Next, we will standardize our parameters using the `scale` function in R:

```
dataset_standardized <- dataset %>%
    mutate(params = scale(params), co2s
     = scale(co2s), week = scale(week),
     dataset_sizes = scale(
    dataset_sizes), train_computes =
    scale(train_computes))

standardized_model <- lm(formula =
    mmlus ~ params + co2s + week +
    dataset_sizes + train_computes,
    data = dataset_standardized)
```

We get a standardized model summary of:

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.2908 -1.3384  0.0847  1.0528  4.1472

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    21.9246     0.1646 133.194  < 2e-16 ***
params         13.3252     3.0941   4.307 3.20e-05 ***
co2s           -4.7266     2.8004  -1.688   0.0938 .
week            5.5811     0.5405  10.325  < 2e-16 ***
dataset_sizes  10.9996     1.3773   7.986 5.80e-13 ***
train_computes -5.9984     0.7864  -7.628 4.08e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.941 on 133 degrees of freedom
Multiple R-squared:  0.9905,    Adjusted R-squared:  0.9902
F-statistic:  2778 on 5 and 133 DF,  p-value: < 2.2e-16
```

Inspired by scaling laws, we introduce log terms and test the fit with the following function in R:

```
log_model <- lm(formula = mmlus ~
    params + co2s + week +
    dataset_sizes + train_computes +
    log(params) + log(co2s) + log(
    dataset_sizes) + log(train_computes
    ), data=dataset)
```

We obtained a subsequent model summary of:

```
Residuals:
    Min      1Q  Median      3Q     Max
-4.7791 -1.4395  0.1151  1.0248  3.9287

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)         -2.297e+02  5.741e+01  -4.002 0.000105 ***
params               4.310e-01  1.126e-01   3.829 0.000200 ***
co2s                -4.270e-01  1.770e-01  -2.412 0.017294 *
week                 1.353e-01  3.931e-02   3.441 0.000780 ***
dataset_sizes        7.366e-12  1.010e-12   7.295 2.70e-11 ***
train_computes      -4.852e-24  5.511e-25  -8.804 7.44e-15 ***
log(params)         -9.207e-01  1.296e+00  -0.710 0.478881
log(co2s)            1.912e-02  9.251e-01   0.021 0.983545
log(dataset_sizes)  -1.713e+00  4.085e-01  -4.192 5.10e-05 ***
log(train_computes)  5.121e+00  1.258e+00   4.069 8.17e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.823 on 129 degrees of freedom
Multiple R-squared:  0.9919,    Adjusted R-squared:  0.9913
F-statistic:  1752 on 9 and 129 DF,  p-value: < 2.2e-16
```

We see that our adjusted R-squared slightly increases to 0.9913, showing that the model fit is slightly better. In addition, when we introduce log terms, we get the following diagnostic plots in Figures 4, 5, 6.

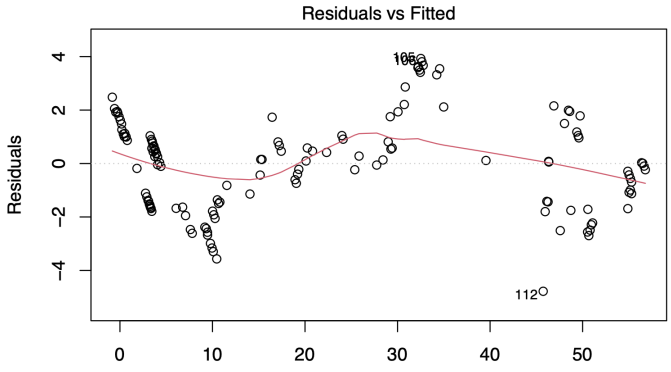


Fig. 7: Residuals vs Fitted Plot

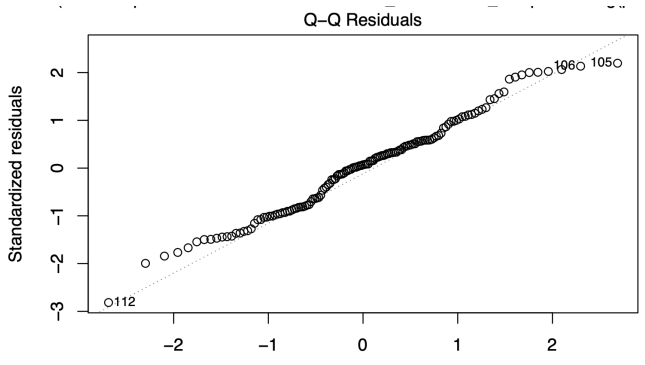


Fig. 8: QQ Plot

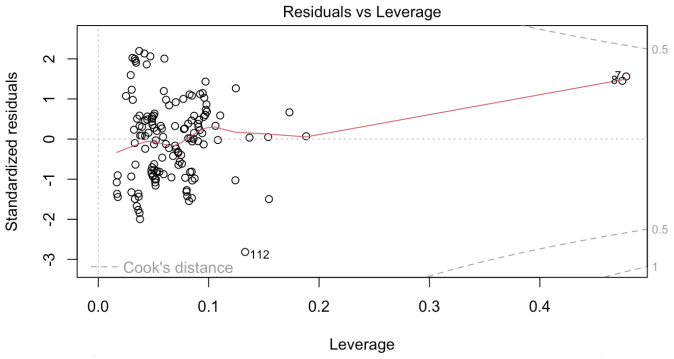


Fig. 9: Residuals vs Leverage Plot

The residuals vs fitted plot and QQ plot are both similar to the baseline model. However, we see that from the central lines that they are less extreme than the plots of the baseline model. Specifically, in the residuals vs fitted plot, the middle line is a little more flat, even though the observations still retain their structure. In the QQ Plot, a similar skew that the baseline model exhibits still exists, but lessens as the extremes hug closer to the theoretical normal line. The residuals vs leverage plot show the extreme observations more sharply, with observations 7, 8, and 112 separated from the rest of the observations. Since we did not collect the data, we will choose to keep the points, but if we were gathering the data, we would more closely inspect the data entries of these outlier observations, considering data mis-entry and fact checking.

We also notice that the negative coefficients in the model summary could be due to high collinearity between the predictors. We will alleviate this by introducing regularization as our main method for predicting LLM model performance.

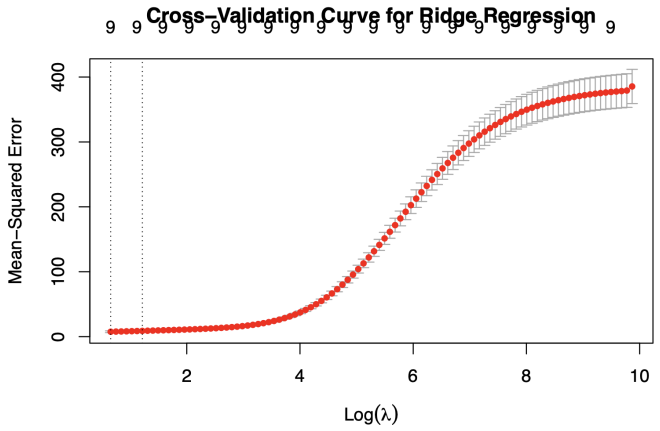
III. Methods and Results

A. Model Transformations

To predict model performance better, we explore leave-one-out cross validation (LOOCV), first trying Ridge regression. First, we perform the cross validation using `cv.glmnet` function in R:

```
X <- model.matrix(mmlus ~ params + co2s
    + week + dataset_sizes +
    train_computes + log(params) + log(
    co2s) + log(dataset_sizes) + log(
    train_computes), da

y <- dataset$mmlus

ridge_loocv <- cv.glmnet(X, y, alpha =
    0, standardize = TRUE, nfolds =
    nrow(X))
```

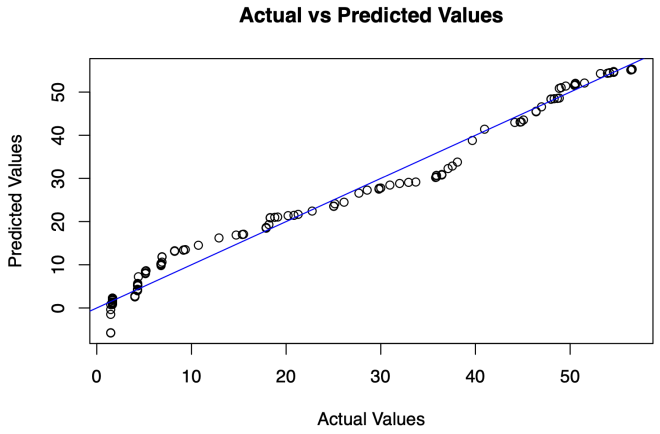
We get a cross-validation curve of:



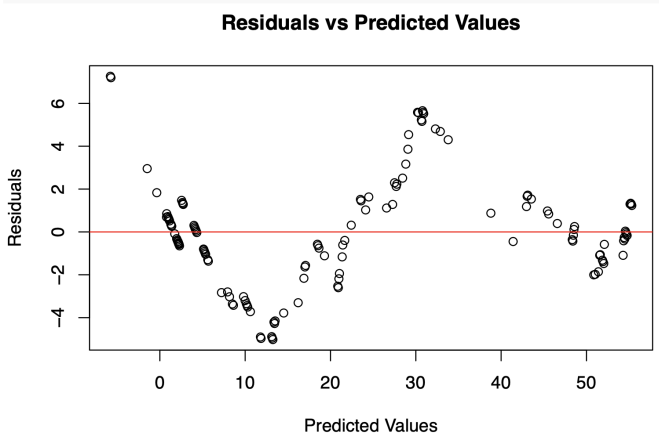
We find that the optimal value of lambda is 1.928765 and that the mean LOOCV error is 7.607066. After fitting the model using the optimal lambda, we get the following model coefficients:

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                s0
## (Intercept)         -8.223104e+01
## params               1.080490e-01
## co2s                 1.806366e-01
## week                 6.679832e-02
## dataset_sizes        2.224878e-12
## train_computes      -2.322291e-25
## log(params)          1.829191e+00
## log(co2s)            6.841229e-01
## log(dataset_sizes)   2.613366e-01
## log(train_computes)  1.352046e+00
```

The actual vs predicted visualization is:



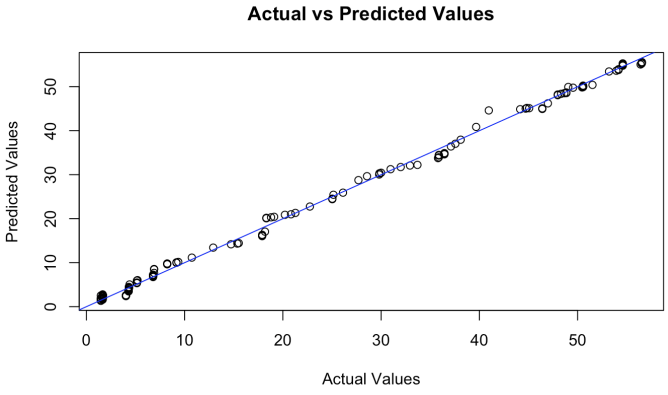
The residual vs predicted visualization is:



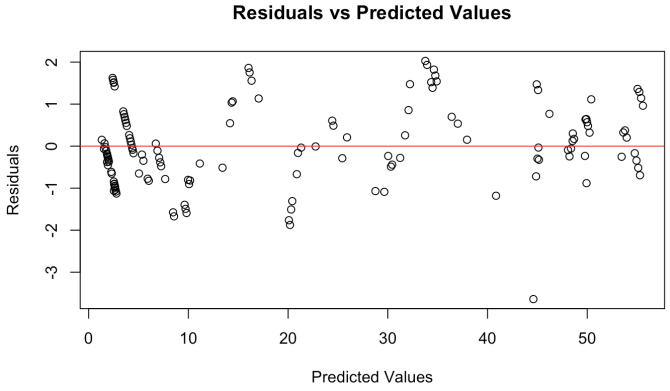
The residual plot above shows that there are polynomial effects. We redo the regularization after introducing polynomial terms with the `poly` function in R:

```
X <- model.matrix(mmlus ~ poly(params
    ,3) + poly(co2s,3) + poly(week,3) +
     poly(dataset_sizes,3) + poly(
    train_computes,3) + poly(log(week)
    ,3) + poly(log(params),3) + poly(
    log(co2s),3) + poly(log(
    dataset_sizes),3) + poly(log(
    train_computes),3), data = dataset)
    [, -1]

y <- dataset$mmlus

ridge_loocv <- cv.glmnet(X, y, alpha =
    0, standardize = TRUE, nfolds =
    nrow(X))
```

We get a similar optimal value of lambda as 1.928765, but a much lower mean LOOCV error of 1.076874. We get an actual vs predicted plot and residual vs predicted plot of:



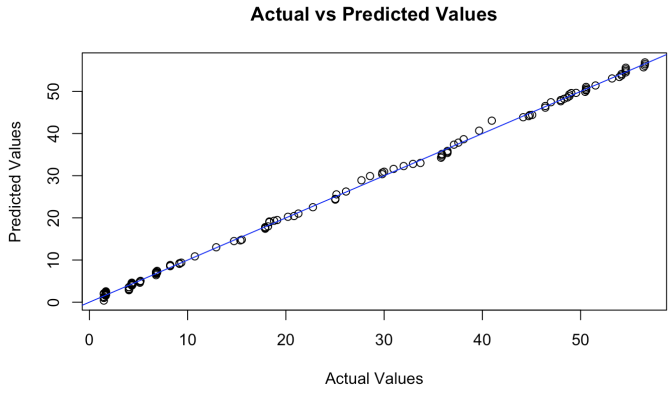
and



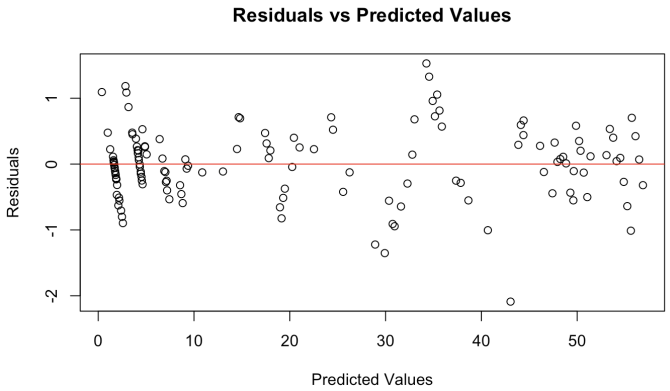
We see after introducing polynomial terms, we get a much better model fit and the residuals are more evenly spread out. Lastly, we will try Lasso regression by setting `alpha = 1` in R:

```
lasso_loocv <- cv.glmnet(X, y, alpha =
    1, standardize = TRUE, nfolds =
    nrow(X))
```

Now, we obtained an optimal value of lambda as 0.01129681, and an even lower mean LOOCV error of 0.5763673. We get an actual vs predicted plot and residual vs predicted plot of:



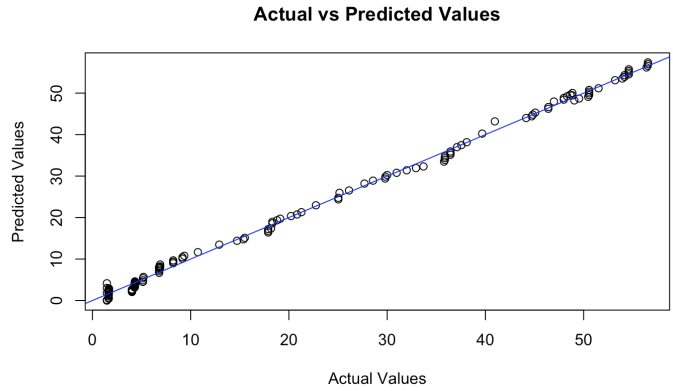
and



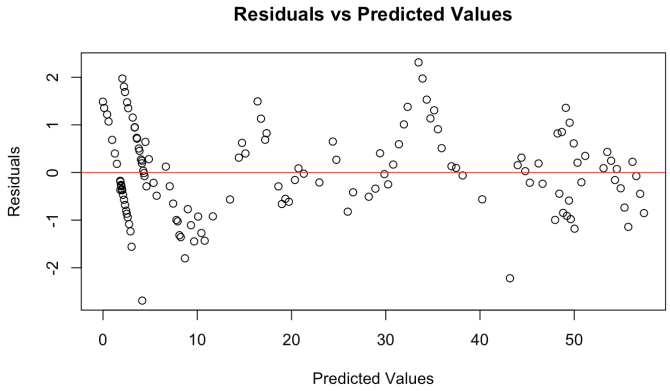
We can see that using L1 norm achieves a better fit than using the L2 norm. Lastly, we will explore if we only used log terms from the previous model to achieve even better fit. Our final model is:

```
X <- model.matrix(mmlus ~ poly(log(week
    ),3) + poly(log(params),3) + poly(
    log(co2s),3) + poly(log(
    dataset_sizes),3) + poly(log(
    train_computes),3), data = dataset)
    [, -1]

y <- dataset$mmlus

lasso_loocv <- cv.glmnet(X, y, alpha =
    1, standardize = TRUE, nfolds =
    nrow(X))
```

We obtained an optimal value of lambda as 0.009976416 , and a mean LOOCV error of 1.356705. We get an actual vs predicted plot and residual vs predicted plot of:



and



We get a LOOCV not far from using polynomial non-log terms, and with much fewer predictors. The coefficients of this final model is:

```
16 x 1 sparse Matrix of class "dgCMatrix"
                                        s0
(Intercept)                      21.9246016
poly(log(week), 3)1             141.7423561
poly(log(week), 3)2              63.5627325
poly(log(week), 3)3              30.1512011
poly(log(params), 3)1            37.5033716
poly(log(params), 3)2            24.5379005
poly(log(params), 3)3           -14.9737645
poly(log(co2s), 3)1             -12.1582904
poly(log(co2s), 3)2               0.8481534
poly(log(co2s), 3)3              10.9249186
poly(log(dataset_sizes), 3)1     53.5179768
poly(log(dataset_sizes), 3)2     29.3841249
poly(log(dataset_sizes), 3)3     12.7706562
poly(log(train_computes), 3)1 -24.0420050
poly(log(train_computes), 3)2  -7.9285082
poly(log(train_computes), 3)3 -14.3665812
```

This model results in an R-squared of 0.9979337 and an adjusted R-squared of 0.9976817, which is greater than the baseline model adjusted R-squared of 0.9902.

B. Model Interpretation

Using these coefficients, we interpret that the higher order log terms are all positive except for the amount of training compute, of which all of its terms have negative coefficients.

Motivated by Chincilla paper, we would expect that the dataset size, parameters and training compute to be highly collinear and only one of these these three terms to be non-zero. The reasoning is that given a compute budget, there is an optimal dataset size and parameter choice to get the best model under the budget constraint. There are two possible reasons to the non-zero coefficients:

1) We have not found the optimal transformation to cancel out the effects by regularization.
2) The model developers did not follow the scaling laws when training their models.

Using the coefficients above, we can interpret that

1) Algorithmic improvements, as proxied by dates, do improve model performance.
2) The number of parameters currently does improve model performance.
3) Test-time compute, as proxied by CO2 emissions for evaluation, does improve model performance.
4) Dataset size does improve model performance.
5) Training compute does not improve model performance, suggesting that developers are using more training compute than necessary.

C. Bonus: Future AI Performance

Since we are extrapolating, this trend might not hold in the future. However, if we assume the trends hold, then we can predict the future predictors via time and then use these predictors to predict future AI progress. First, we predict the future predictors in R:

```
model_param <- lm(log(params) ~ log(
    week), dataset)
model_co2 <- lm(log(co2s) ~ log(week),
    dataset)
model_dataset_size <- lm(log(
    dataset_sizes) ~ log(week), dataset
    )
model_train_compute <- lm(log(
    train_computes) ~ log(week),
    dataset)

weeks_then <- as.integer((as.Date
    ("2025-06-01")-start_date)/7)
new_param <- predict(model_param, data.
    frame(week = weeks_then))
new_co2 <- predict(model_co2, data.
    frame(week = weeks_then))
new_dataset_size <- predict(
    model_dataset_size, data.frame(week
     = weeks_then))
new_train_compute <- predict(
    model_train_compute, data.frame(
    week = weeks_then))
```

Next, we predict half a year into the future of June 1, 2025, in R:

```
new_df <- data.frame(
  params = new_param,
  co2s = new_co2,
  week = weeks_then,
  dataset_sizes = new_dataset_size,
  train_computes = new_train_compute,
  mmlus = -100
)

X <- model.matrix(mmlus ~ poly(log(week
    ), 3, raw = TRUE) + poly(log(params
    ), 3, raw = TRUE) + poly(log(co2s),
```

```
    3, raw = TRUE) + poly(log(
    dataset_sizes), 3, raw = TRUE) +
    poly(log(train_computes), 3, raw =
    TRUE), data = new_df)[, -1]

new_mmlu <- predict(lasso_model, s =
    optimal_lambda, newx = X)
```

We get a predicted MMLU of 203.9229.

The maximum theoretical MMLU score is 100, but one way we can read this is that the open-weight AI models should plateau before reaching 100, or they will indeed score a 100 on the MMLU before the middle of next year.

IV. Conclusion and Discussions

This project set out to explore the factors driving the progress of large language models (LLMs), specifically focusing on how algorithmic improvements, training compute, dataset size, and the number of parameters influence model performance. After performing our initial Exploratory Data Analysis, we found the number of valid observations was extremely limited, as mentioned in the meeting with our assigned TF, Gabe. Consequently, we found a way to increase the number of observations by considering the state-of-the-art of the predictors to predict the state-of-the-art performance. Essentially, we subset the data by weeks instead of by models, increasing the number of observations to 139! After applying a series of transformations, such as applying log terms and leave-one-out cross-validation, we arrive at an adjusted R-squared of 0.9977, which means that our model explains about 99.77% of the variance in the dependent variable, the MMLU score. Specifically, the model contains the following important variables with our findings:

1) **Algorithmic Improvements:** As proxied by the publication date of models, we observed that algorithmic improvements do indeed contribute to improved model performance. More recent models generally performed better on benchmarks, supporting the idea that continual innovation drives progress in LLM capabilities.

2) **Model Parameters:** We found that increasing the number of parameters in models has a positive association with better performance. The supports the idea that bigger models can hold more computational complexity and capture the variability in the data.

3) **Test-time compute:** Our analysis revealed that the amount of compute used during model test-time, which we approximated through CO2 emissions, is positively associated with model performance. This suggests that investing in greater computational resources tends to improve results.

4) **Dataset Size:** Larger training datasets are consistently associated with improved model performance, aligning with the hypothesis that more diverse and extensive data enables LLMs to better generalize and perform on various tasks.

5) **Training Compute:** We found that (keeping all else equal) increased training compute was associated with decreased model performance, which is surprising, but we can reconcile this with the reframing of the problem statement. With the reframing of how well do state-of-the-art predictors predict state-of-the-art model performance, we can reason that developers are using more training compute than necessary, even though training compute generally improves model performance.

Looking forward, the trends identified in this analysis suggest that LLM progress is largely driven by a combination of algorithmic improvements, compute, and dataset size, with diminishing returns for very large models or training compute. By predicting future trends in these factors, we project that AI models will continue improving but will face diminishing returns in terms of performance as they approach optimal configurations of parameters and compute.

Future work could focus on refining these models by incorporating additional data, adjusting for more refined collinearity, and exploring alternative performance benchmarks. Additionally, experiments on the cost-efficiency of these models could provide valuable insights for AI development moving forward.

APPENDIX

The datasets, Rmd file, and knitted R PDF can be found in this folder.