Inferring Neural Network Scaling Laws from Compute Market Data

Ian Joffe

ECMA 35050, Autumn 2024

1 Introduction

The rise of deep learning over the past decade, and especially in the past 3 years, can be attributed in part to technical and methodological advances. But perhaps more importantly, deep neural networks have benefited from ever-increasing scale in the amount of compute used to train them. Large AI labs have put faith in training more and more massive models, on more and more data, often betting millions of dollars that a larger training run will yield a better LLM, computer vision model, chess player, or decision-maker in any domain. These bets are not without justification. Those who have observed the training of thousands of models, often systematically, have calculated rules about how model performance is expected to increase with model size. If these *scaling laws* hold, they give AI labs the power to predict their model's abilities before it's trained.

Those with the resources may be able to run enough experiments to see the scaling laws materialize for themselves. But as frontier-level models become prohibitively expensive to train – and the frontier is where questions over the scaling laws become the most enthralling – those of us without supercomputer clusters may be left behind. I believe a solution to this is to collect data from *compute markets*. In a compute market traders, including those wanting to build massive models, buy and sell time on the GPU computers used for model training. The market may also be used by individuals using GPUs for other tasks, so getting information out of such markets requires separating the signal from the noise. This project sets up a plan for doing so.

2 Literature Review

2.1 The Batch Trading Model

The batch trading model was introduced by Kyle ([1], 1985) to model situations where many traders' beliefs are revealed at the same time. Consider an asset with some true value. Kyle's model assumes that some noisy version of that true value is known to the public. Informed traders have a more accurate (or potentially perfectly accurate) signal about the true value which they use to trade. Uninformed traders trade based on an even noisier version of the public estimate of the true value. An observer gets to see a large number of trades, but doesn't know which are informed and which are uninformed. Their objective is to estimate the asset's true value. Kyle was specifically interested in the case where the dealer is the one estimating the true value, and using it to price trades.

I avoid introducing notation here because I won't be using Kyle's exact model, but the theme will be the same. I assume that many participants contribute to a batch of orders, some of whom have useful information and others of whom are trading with a lot of noise. The objective is to extract the useful information from the full batch, without being told who's who.

2.2 Scaling Laws

The relationship between model performance, model size, and data size is at the core of the field of statistics. Model performance is always measured with test loss L, that is how low it can get its objective function on data that it did not access during training. A scaling law is an equation that gives model performance as a

function of the number of data points it sees, D, and the number of learnable parameters it contains N. In the context of LLMs, the number of data points is the number of tokens it sees (assuming on epoch of training), and the parameters are contained in learned weight matrices for each layer. Scaling laws should generalizes well. A good scaling law should accurately interpolate the performance of models with size in between those used to fit the law, which span several orders of magnitude, and a great scaling law should extrapolate to even larger models, in data and in parameters. A prefect scaling law, which is truly a *law*, can even extrapolate even to new hyperparameters and model tweaks that it was not fit to and may yet to be invented.

The first paper to attempt to write scaling laws for modern neural networks was Hestness et al. (2017 [2]), who noticed that given a minimum for N and D is surpassed, neural net scaling followed a power law of form

$$L = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}. \tag{1}$$

E represents irreducible error due to the entropy in the data. The other parameters: A, B, α, and β, show how model loss responds to changes in N or D.

Hestness et al. made an attempt to estimate these parameters, but the first widely accepted fit came from Kaplan et al. (2020, [3]), who trained a diverse set of models to estimate that $0 < \alpha < \beta < 1$. Furthermore, they applied this result to make a claim about the optimal values of N and D if total compute C is constrained. Using the approximation

$$C = 6ND \tag{2}$$

which is justified by the details of neural network architecture, Kaplan et al. claim that compute should be allocated in a way that scales D faster than N. The exact equations, derived from the power law, are

$$N_{opt} = \left(\frac{\alpha A}{\beta B}\right)^{1/(\alpha+\beta)} \left(\frac{C}{6}\right)^{\beta/(\alpha+\beta)} \qquad D_{opt} = \left(\frac{\beta B}{\alpha A}\right)^{1/(\alpha+\beta)} \left(\frac{C}{6}\right)^{\alpha/(\alpha+\beta)}. \tag{3}$$

Two years later, Hoffman et. al (2022, [4]) published a paper suggesting that Kaplan didn't sufficiently explore the space of possible N and D and overlooked training techniques leading their models to be underfit. They, too, train a diverse array of neural networks and instead propose $\alpha \approx \beta \approx 0.5$. This implies that compute should be allocated to scale N and D at the same rate. At least in public discourse, Hoffman et al.'s findings are still considered to hold up today, even after two more years of model innovation and orders of magnitude of increase in C.

More recently for LLMs, the scaling law paradigm has been applied to a new variable: inference tokens I. When an LLM is asked a question, it might try to guess the answer immediately, or it might write out a "chain of thoughts," reflecting on and evaluating its own output until it decides that its answer is satisfactory. Each output token requires extra compute, so an AI lab who has limited compute to allocate between training model and serving lengthy chain-of-thought responses to customers might like to quantify this tradeoff. With the release of OpenAI's chain-of-thought model o1, the team released graphs (2024, [5]) claiming that inference, too, followed a power law

$$L = E + \frac{G}{F^{\gamma}} \tag{4}$$

where F is a measure of the compute used during inference, such as the number of tokens outputted, and G and γ are the power law's parameters. There is not yet public consensus on the values of G and γ, or how they relate to α and β, which makes how to allocate compute between training and inference an open problem.

There are hundreds of papers on scaling laws, but all I could identify perform estimation by doing a large number of training runs themselves, about which they have full information. My approach in this project is unique in not only assuming very limited information about the runs, but also setting up an environment where they're performed at many different labs. While this approach does lose some information on each run, it has the advantage of aggregating the scales, techniques, errors, and other minutia of all labs who participate in a market, even when those techniques are kept secret. This may have the positive effect of making the estimated laws more robust to new architectural details not included in the models it was fitted to.

2.3 Compute Markets

Cloud computing, a practice in which a large provider of compute rents it to other users, has now been around for multiple decades. AWS, for instance, is 22 years old. Cloud providers offer a large menu of products and prices to fit all kinds of business needs, and even users of the same cloud product may have diverse use cases. There's a good amount of literature on general cloud markets: Sharghivand et al. (2016, [6]) provide a survey of work on cloud compute auctions in particular. A zoo of complex, and no doubt advantageous mechanisms have been proposed, although today's largest providers still mostly rent compute for fixed prices, which I expect are also based on extensive theory.

GPUs, while originally created for graphics-related workloads, have recently been used for more many more general applications including cryptocurrency mining and, most relevant to this project, AI training and inference. Many of the traditional cloud providers and a slew of new companies have created business renting out GPU compute, especially to those looking to use AI. However, I could not find fresh public research has been done on how GPU markets may differ from traditional cloud markets. Given that most buyers of cloud GPU are using it for AI (such participants have proven willing to outbid gamers and crypto enthusiasts), sellers may be able to take advantage of patterns in their behavior to better optimize pricing, and other observers may also be able to extract information they find interesting from GPU sale data.

The traditional cloud providers have refrained from giving their resources to open markets or allowing buyers to resell GPU, opting to set their prices however they choose. Other companies, however, have tried to implement market mechanisms. Vast AI [7] allows anyone to be a host by downloading Vast's software and listing their GPUs for rent. Hosts set a max price, which any buyer may pay to get remote access to the GPU immediately. Hosts may also set a min price. If no one is willing to pay the max price, then buyers can submit bids and the highest bidder who bids at least the min price will get access to the GPU until someone outbids them or pays the max price.

Vast's approach is to welcome all kinds of hosts with diverse GPU setups and other equipment. Another company, San Francisco Compute [8], takes a different approach. Only one type of GPU, Nvidia's H100, is available. Hosts must pass a careful audit to participate. But once a host is added to SF Compute's market, the host becomes a seller of the hours on that GPU, and all GPU hours on the market can be bought or sold with market or limit orders. This setup makes SF Compute's market easier to analyze than Vast's, so it'll roughly guide how I set up my theory.

It's worth noting that other attempts have been made at more complex GPU market schemes using blockchain technology. One company which has had some success is Akash Network, whose whitepaper (2020, [9]) details how a proof-of-stake based blockchain allows for a transparent and verifiable order book, along with flexibility for how different users of the chain charge fees. Furthermore, distribution of tokens whose value may inflate if the market becomes popular to early GPU providers incentivizes providers to sell within their chain. Paul Timofeev (2024, [10]) describes a number of other active blockchain-based compute markets ("DePINs"), each with their own bells and whistles. But from a simplified perspective, while the blockchain allows for a myriad of incentivization schemes, all of these models come down to different kinds of traders interacting through an order book.

2.4 Related Markets

While there's limited academic research on GPU markets in particular, there's plenty of work on similar commodity markets. In a commodity market, the assets traded represent ownership of something physical. GPU markets are particularly similar to electricity markets, which are detailed in Glachant et al.'s textbook (2021, [11]). One similarity is that buyers and sellers of compute or electricity typically don't match in the volume they're trying to buy or sell. So, it's the market's job to not only batch buyers and sellers, but to coordinate between them accounting for all resources. Another similarity is that failure of the underlying asset is common. If one buys an asset giving them access to gold, they should be able to count on that gold working just like any other gold in the world. But when you buy a large enough number of GPUs, some of them are bound to break, just like how if you buy enough electricity, some of the transmission lines will

inevitably fail. Markets account for this in different ways - some leave it at "too bad, that's the game," while some offer refunds at the expense of the market, some at the expense of the provider, and some in the form of replacement compute or electricity instead of money. A third similarity is that physical locality matters. In electric grids, having your power source closer to you means you'll be able to transmit it faster. In compute markets, it's less important how close the GPUs are to you, but more important how close they are to each other. If you require more compute than any one provider can provide, buying half your GPUs in North America and the other half in Asia is a bad idea because whenever they have to communicate with each other, you'll experience high latency. So, a buyer's utility is not strictly minmizing price - they also have to include some cost function of the location of their machines. I'll abstract over most of these details in this project, but they're interesting to note, and would make for great future extensions.

So far, we've only mentioned two types of compute buyers: those who want to run long training runs, and those who want to run all kinds of more diverse experiments. But, there's also a third kind: speculative buyers. These buyers don't intend to use the GPU at all, but instead hope to flip their share of it for more than they bought it for. While many markets may be comprised of mostly speculative traders, this is risky in a compute market because if you're stuck with the GPU's when you're time with them begins, and your time on them starts to pass, the value of your asset goes to zero (or some small quantity from whatever crypto you can mine with them, less than an AI lab would pay to use them). Many commodity markets involve similar risks - consider the corn futures trader who got stuck with a ton of corn, and has to keep it around the house since he has nowhere else to put it. But electricity markets are especially similar since the value of your time on the grid also goes to zero after that time passes. Such phenomena are best modeled using stochastic processes, similar to options trading. That's out of the scope of this project, but I'll briefly address a simpler model later.

3 An Independent Observer's Model

3.1 Setup of Buyers

We consider a market with two kinds of buyers. One is AI labs who use a lot of GPU for large training runs. Consistent with the literature, we call these the informed traders, as we assume they have some knowledge of the correct scaling laws. The other is all other companies and individuals buying GPUs for small-sacle experiments or other activities, called the noise traders. Recall that we're interested in estimating the parameters of the scaling laws. Since these buyers aren't looking to train massive models, they won't provide us with information about scaling. There will be n_{total} total orders, with n_{informed} from the labs and n_{noise} from the noise buyers. n_{total} is known, while n_{informed} and n_{noise} are unknown.

To begin, we don't consider the cost at which compute is purchased, and just assume that we see a full batch of fulfilled orders, some of which are from informed traders and some of which are from noise traders. Specifically, we view the total amount of compute in each order $C_1, C_2, ..., C_{n_{\text{total}}}$.

The scaling law parameters we're interested in estimating are E, A, α, B, and β. To do this, we'll make some initial assumptions:

Assumption 1. The amount of compute that AI labs purchase follow a lognormal distribution $\text{LogN}(\mu_{\text{inf}} + \log\sqrt{6}, \sigma^2_{inf})$. The amount of compute that other buyers purchase follows a lognormal distribution. $\text{LogN}(\mu_{\text{uninf}}, \sigma^2_{uninf})$. The C_i's drawn from thse distributions are independent.

Assumption 2. AI labs always have selected N_i and D_i, that is the amount of data and number of parameters for their model that run i is for, to optimize loss given the amount of compute they bought.

I do not have the data to justify assumption 1 empirically, but the lognormal is useful for three reasons. First, its support is nonnegative. Second, logarithms play nicely with power laws. Third, the product of two lognormals is another lognormal. So by Equation (2), we may have $N_i \sim \text{LogN}(\mu_N, \sigma^2_N)$ and $D_i \sim \text{LogN}(\mu_D, \sigma^2_D)$. The factor of 6 in Equation (2) necessitates the extra term in the mean of C_i's distribution,

but since $\log\sqrt{6} = 0.39$ is generally much smaller than μ_{inf}, it can safely be ignored. Of course, when the ith sample is not actually from an informed trader, these values don't mean anything. I have some regrets about using the lognormal. I thought that because the tails were subexponential they might be thick enough, but having looked at more simulations since I made this decision, I don't believe they are, which is important since the C_i's span many orders of magnitude. In summary, the data is our model is assumed to be distributed as

$$C_{\text{uninf}} \sim \text{logN}(\mu_{\text{uninf}}, \sigma^2_{uninf}) \qquad N_i \sim \text{LogN}(\mu_N, \sigma^2_N) \qquad D_i \sim \text{LogN}(\mu_D, \sigma^2_D) \tag{5}$$

The second assumption is reasonable because our premise is that the AI labs are well-informed about the scaling laws. However, it is possible to incorporate uncertainty or disagreement about the scaling laws among the labs, which I describe in Section (5.2).

3.2 Scaling Law Estimation

Let f_N be the lognormal density with parameters μ_N and σ^2_N, let f_D be the lognormal density with parameters μ_D and σ^2_D, and let f_{uninf} be the lognormal density with parameters μ_{uninf} and σ_{uninf}. We introduce n_{total} latent Bernoulli random variables $I_1, I_2, ..., I_{n_{\text{total}}}$ which are 1 when the ith compute order is by a lab, and 0 when the ith compute order is by a noise trader. Applying Assumption (2) and plugging in the formulas for N_{opt} and D_{opt},

$$P(C_1, ..., C_{n_{\text{total}}}) = \prod_{i=1}^{n_{\text{total}}} I_i f_N\left(\left(\frac{\alpha A}{\beta B}\right)^{1/(\alpha+\beta)}\left(\frac{C_i}{6}\right)^{\beta/(\alpha+\beta)};\ \mu_N, \sigma^2_N\right) f_D\left(\left(\frac{\beta B}{\alpha A}\right)^{1/(\alpha+\beta)}\left(\frac{C_i}{6}\right)^{\alpha/(\alpha+\beta)};\ \mu_D, \sigma^2_D\right)$$
$$+ (1 - I_i) f_{\text{uninf}}\left(C_i;\ \mu_{\text{uninf}}, \sigma^2_{\text{uninf}}\right) \tag{6}$$

The EM algorithm (1977, [12]) can be applied to maximize the (log of the) likelihood above to estimate A, α, B, and β. Information about loss would be required to estimate E. You'll get the rest of the parameters and the I_i's as well. In the E-step, you'll set

$$\begin{cases} I_i = 1 \text{ if } f_N(\cdots) f_D(\cdots) \geq f_{\text{uninf}}(\cdots) \\ I_i = 0 \text{ if } f_{\text{uninf}}(\cdots) > f_N(\cdots) f_D(\cdots). \end{cases} \tag{7}$$

It may be helpful to assume that only a small number of traders are AI labs doing training runs. This can be accomplished by setting a Bayesian prior $p \in (0, 1)$ that any given order is from a lab. Then the E-step looks like

$$\begin{cases} I_i = 1 \text{ if } p\, f_N(\cdots) f_D(\cdots) \geq f_{\text{uninf}}(\cdots) \\ I_i = 0 \text{ if } (1-p)\, f_{\text{uninf}}(\cdots) > f_N(\cdots) f_D(\cdots). \end{cases} \tag{8}$$

For convenience, let $\hat{n}_{\text{inf}} = \sum_{i=1}^{n_{\text{total}}} I_i$ and let $\hat{n}_{\text{uninf}} = \sum_{i=1}^{n_{\text{total}}} (1 - I_i)$. To perform the M-step, I go through some tedious calculus to maximize the log likelihood with respect to each parameter. I was able to mostly

solve for closed form solutions

$$
\begin{aligned}
A &= \frac{\beta}{\alpha} B \exp\Big(\Big(\sigma_N^2 - \sigma_D^2\Big)\Big(\frac{\mu_N}{\sigma_N^2} - \frac{\mu_D}{\sigma_D^2} + \frac{1}{\hat{n}_{\text{inf}}}(\frac{\alpha}{\sigma_D^2} - \frac{\beta}{\sigma_N^2})\frac{1}{\alpha+\beta}\sum_{i:\, I_i=1}\log\frac{C_i}{6}\Big)\Big) \\
B &= \frac{\alpha}{\beta} A \exp\Big(\Big(\sigma_D^2 - \sigma_N^2\Big)\Big(\frac{\mu_D}{\sigma_D^2} - \frac{\mu_N}{\sigma_N^2} + \frac{1}{\hat{n}_{\text{inf}}}(\frac{\beta}{\sigma_N^2} - \frac{\alpha}{\sigma_D^2})\frac{1}{\alpha+\beta}\sum_{i:\, I_i=1}\log\frac{C_i}{6}\Big)\Big) \\
\mu_N &= \frac{1}{\hat{n}_{\text{inf}}}\sum_{i:\, I_i=1}\Big(\frac{1}{\alpha+\beta}(\log\frac{\alpha A}{\beta B}) + \frac{\beta}{\alpha+\beta}\log\frac{C_i}{6}\Big) \\
\sigma_N^2 &= \frac{1}{\hat{n}_{\text{inf}}}\sum_{i:\, I_i=1}\Big(\frac{1}{\alpha+\beta}(\log\frac{\alpha A}{\beta B}) + \frac{\beta}{\alpha+\beta}\log\frac{C_i}{6} - \mu_N\Big)^2 \\
\mu_D &= \frac{1}{\hat{n}_{\text{inf}}}\sum_{i:\, I_i=1}\Big(\frac{1}{\alpha+\beta}(\log\frac{\beta B}{\alpha A}) + \frac{\alpha}{\alpha+\beta}\log\frac{C_i}{6}\Big) \\
\sigma_D^2 &= \frac{1}{\hat{n}_{\text{inf}}}\sum_{i:\, I_i=1}\Big(\frac{1}{\alpha+\beta}(\log\frac{\beta B}{\alpha A}) + \frac{\alpha}{\alpha+\beta}\log\frac{C_i}{6} - \mu_D\Big)^2 \\
\mu_{\text{unif}} &= \frac{1}{\hat{n}_{\text{uninf}}}\sum_{i:\, I_i=0}\log C_i \\
\sigma_{\text{uninf}}^2 &= \frac{1}{\hat{n}_{\text{uninf}}}\sum_{i:\, I_i=0}(\log C_i - \mu_{\text{uninf}})^2 \\
p &= \frac{\hat{n}_{\text{inf}}}{n_{\text{total}}} \quad \text{(if used)}
\end{aligned}
\tag{9}
$$

Unfortunately, the equations for the most important parameters, α and β, were miserably long, and I suspect don't have a closed-form solution. But you can absolutely get expressions including them by differentiating the log likelihood, and one could plug them into your favorite solver to complete the M-step. So we have arrived upon an an algorithm to, given our assumptions, successful estimate the scaling law parameters.

Note that solutions to the optimization problem are only identifiable up to a symmetry between $(A, \alpha, \mu_N, \sigma_N^2)$ and $(B, \beta, \mu_D, \sigma_D^2)$. A solution where the values of those tuples are switched is always equally likely. One way to break this symmetry is by picking a known value for any one of the eight parameters, and fixing it throughout the procedure. Alternatively, you could assume some inequality between any corresponding pair, for instance α and β, and use that to identify whether you should flip the values of the tuples after EM. If you prefer to make no further assumptions at all, this method is still useful for testing equalities between the symmetric parameters, for example Hoffman's approximate $\alpha = \beta$. One way to perform this test would be to bootstrap the C_i's, and calculate the corresponding parameter values for each bootstrapped sample.

4 A Seller's Model

In Section (3) we considered a nonreciporical perspective, where we consider the C_i's to be given, either because the market dynamics have already been worked out or because informed buyers just purchase however much compute they can afford (in that case, their budget is lognormally distributed). Here, we consider the case where the compute is currently owned by sellers, and the sellers use a previous batch of orders to determine how much to charge for the next batch. This is closer to Kyle's original batch model, and provides new insights.

In GPU markets we may assume that each unit of compute has the same price q because sellers want to be able to offer the same GPU to the highest bidder, which requires putting all bidders on an equal playing field, regardless of how many GPU's they've already bought. In other industries bulk deals may help sellers get rid of their inventory, but demand for GPUs is so high that this is generally not a concern.

Recalling a trained AI model's loss $L(N, D)$ measures its performance, a buyer will have a utility function $u(L)$. Different labs will have different u's based on their objectives. Those with the primary goal of developing AGI, which is expected to only emerge with very high compute, might even have convex utility functions and only be bounded by their total budget. Or more accurately, they may have a sigmoidal utility function, where they don't get much utility from small less intelligent models, but also get decreasing returns once models can already perform tasks as well as can be evaluated. Other labs might be more focused on developing models that do individual tasks very well. In this they only need a little compute for each training run, and don't get much utility for the extra compute that will let the model generalize to other tasks. So their loss functions would be concave, as we're more used to.

Since demand is very high, buyers generally shouldn't expect extra utility for their money. So assuming linear utility of money, we can find the price q_u that buyers with utility u are willing to pay by equating the price of compute and utility of loss, putting together Equations (1), (2), and (3). This again assumes that labs select the optimal N and D for their compute budget.

$$\begin{aligned} q_u C &= u(L) \\ &= u(E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}) \\ &= u\bigg(E + (\frac{C}{6})^{\alpha\beta/(\alpha+\beta)}\Big(A(\frac{\alpha A}{\beta B})^{-\alpha/(\alpha+\beta)} + B(\frac{\beta B}{\alpha A})^{-\beta/(\alpha+\beta)}\Big)\bigg) \\ q_u &= \frac{1}{C} \cdot u\bigg(E + (\frac{C}{6})^{\alpha\beta/(\alpha+\beta)}\Big(A(\frac{\alpha A}{\beta B})^{-\alpha/(\alpha+\beta)} + B(\frac{\beta B}{\alpha A})^{-\beta/(\alpha+\beta)}\Big)\bigg) \end{aligned} \tag{10}$$

If u is logarithmic and E is sufficiently small (i.e. your data is expected to be predictable with a massive enough model) then we can make some minor simplifications, but can't get too far. We can write

$$q_u \approx \frac{\alpha\beta}{\alpha+\beta}\frac{\log C}{C} + \mathcal{O}(\frac{1}{C}). \tag{11}$$

When C is large, the first term dominates so we can say that under such conditions, buyers are willing to pay more per unit of compute when α and β are close together. Note that log utility is especially concave, so this applies more closely when labs are happy to train small models and uninformed traders also get heavily decreasing returns. Other utility functions will have even more complicated relationships, and in the most realistic case of a distribution of utility functions, interpretation is especially difficult.

For the seller to maximize their expected payout, they'll select q to be

$$\begin{aligned} q_u^* = \text{argmax}_q\ qE(C_{\text{sold}}) &= \text{argmax}_q \int C \cdot \mathbf{1}_{q \le q_u(C)}\ dP(C) \\ &= \text{argmax}_q \int_C C \cdot \mathbf{1}_{q \le q_u(C)}(p f_N(N_{opt}) f_D(D_{opt}) + (1-p) f_C(C))\ dC. \end{aligned} \tag{12}$$

$q_u(C)$ is given by Equation (10), $\mathbf{1}$ is the indicator function, N_{opt} and D_{opt} are given from C in Equation (3), and p is estimated probability that an individual who comes across and wants to purchase compute is a big lab. It's hard to see this integral being solvable in closed form, even with convenient utility functions like log, so numerical methods will have to be applied.

Equation (12) make a lot of assumptions. Not only does it carry over Assumption (1) and Assumption (2) from Section (3), but it also supposes that all labs *and all uninformed participants* have the same utility function, and that the seller's supply of GPU is unlimited. It's especially unrealistic to assume that utility works the same way for AI labs and noise traders, but maybe to some this model is a useful starting point.

So far in this project, including Equation (12), we take the angle that buyers come around wanting to purchase a specific amount of compute (perhaps they're only interested in their model can hit a specific loss), and either buy it or not. A different perspective would be to assume that every lab buys the optimal

amount of compute for their utility function. This perspective is more realistic in that labs are usually trying to optimize a continuous utility (they'd rather have a mediocre LLM than none at all), but it fails when an extremely AGI-hungry lab comes along with convex utility, as they would purchase infinite compute. This issue can be resolved by placing a hard cap on the amount of compute that any one lab can buy, but for simplicity, I'll assume concave u. In this setup, the buyer simply solves

$$q_u^* = \text{argmax}_q \; qC_u(q) \tag{13}$$

where $C_u(q)$ is the inverse of Equation (10). For nontrivial u's, I expect the inversion to require numerial approximation. Equation (12) still suffers from the same assumption as (11), that all buyers share the same utility function, although it's a bit easier in this case to try to integrate over a distribution of utility functions, if interested. This would be instead of an integration of C, since we've now let C be fixed based on q instead of drawing it from a distribution.

Of course calculating Equation (10), which is necessary for (11) and (13), requires the seller to estimate the scaling law parameters. In the random C case of Equation (11), we did this in Section (3). The back-and-forth between parameter estimation and pricing introduces interesting dynamics, which I begin to address in Section (5.4). In the fixed C case of Equation (13), if u is also fixed, the seller will have to rely on previous knowledge of scaling laws or experiment with other options for q while they optimize. This experimentation is related to but distinct from Section (5.4), and would would make for a fascinating follow-up to this project. Alternatively, if u is a assumed to follow a distribution, an EM algorithm similar to that introduced in Section (3) could be developed based on observations.

5 Potential Further Extensions

5.1 Incorporating More Kinds of Buyers

5.1.1 Large Labs vs. Small Labs

In Section (3.1), the latent I_i's describe what distribution the C_i's are drawn from. We established one distribution for AI labs, and another for noise buyers. It might better describe the data to include more latents, which are one-hot. For example, we might have three latents for each data point: $I_1 = 1$ if the order was from a large AI lab, $I_2 = 1$ if the order was from a small AI lab, and $I_3 = 1$ if the order was from a noise trader. This allows us to fit a distribution with a different mean, or even a different distribution altogether, for the large and small labs. Such a setup may better fit the data if AI labs' orders are indeed multimodal. At the risk of overfitting, you're free to further increase the number of latents per data point to describe more types of labs or noise traders. Note that for EM with one-hot latents, the E-step should only calculate expectations for all but one of them, and the last one's expectation is set to the remaining probability.

5.1.2 Speculative Buyers

In Section (2.4), I briefly introduced buyers who are interested in purcahasing compute only because they think they can resell it for a higher price. Speculative traders like this don't fit perfectly into the batch model because understanding them requires temporal information about trading, whereas the batch model just includes one dump of all trades. But even leaving incorporation of time to a follow-up project, we can begin to think about how, like in Section (5.1.1), the dealer might have a one-hot latent for each of labs, noise traders, and speculative traders, when trying to determine who made a trade. While those who we guess are noise traders are ignored in estimation of scaling law parameters, those who we guess are speculative traders may have some impact based on our assumptions about their private knowledge. Their role in the formulas will be in addition to, but different from the labs.

5.1.3 Buyers for Inference

GPU is necessary both to train models and to use them to produce outputs on new data, a process called inference. As I discuss in Section (2.2), an increasing share of recent GPU use has been for inference. Again,

like in (5.1.1), we can introduce a new latent to assign orders in the data to inference buyers. If we assume a distribution on inference purchases, as we did for training purchases, we can estimate γ and G from Equation (4) in each step of EM very similarly to Equation (9).

A more interesting problem arises when considering that the same buyers may want to buy both compute for training and compute for inference. If they're hoping to minimize total loss across one training run but m inference runs, their objective is

$$\min_{\substack{C,F_1,...,F_j \text{ s.t.} \\ C+F_1,...,F_1=C_{\text{total}}}} L(N_{opt}(C), D_{opt}(C), F_1, ..., F_m) = \min_{\substack{C,F_1,...,F_m \text{ s.t.} \\ C+F_1,...,F_1=C_{\text{total}}}} m \cdot (E + \frac{A}{N_{opt}(C)^\alpha} + \frac{B}{D_{opt}(C)^\beta}) + \sum_{j=1}^{m} \frac{G}{F_j^\gamma} \tag{14}$$

To estimate scaling laws in the same way as Section (3.2), the one can assume that the buyer has solved this constrained optimization problem to determine how any amount of C_{total} should be split between training and inference, and then that the distribution which used to be on C is now on C_{total}. A simplifying assumption to first approach the optimization would be that $F_1 = F_2 = ... = F_m$, that is all inference runs use the same amount of compute. A more advanced approach would consider that different inference runs may require different compute F, so it would model each F_j with its own distribution with parameters that are a function of C_{total} instead of assuming it to be a fixed function of C_{total}. A rigorous setup for this problem is unfortunately out of scope of this project.

5.2 Labs Don't Know the Exact Scaling Laws

In the model described in Section (3.1), it's assumed that labs always select the optimal values for N and D from C, which they're able to do because they have perfect knowledge of the scaling laws. This is unrealistic, since while some progress has been made, we don't yet have a general enough understanding of their theoretical backing. Different labs have different emprical evidence, and different interpretations of the empirical evidence. We can introduce uncertainty over the scaling laws by introducing additional latents to Equation (6) which represent such noise. For example, if wanted to assume that the lab making order C_i (if it is an informed order) believes that α is the true value of α plus noise $Z_i \sim N(0, \sigma_\alpha^2)$, we could rewrite the $f_N(\cdot)$ term in Equation (6) as

$$f_N\left(\left(\frac{(\alpha + Z_i)A}{\beta B}\right)^{1/((\alpha+Z_i)+\beta)} \left(\frac{C_i}{6}\right)^{\beta/((\alpha+Z_i)+\beta)};\ \mu_N, \sigma_N^2\right) \phi\left(Z_i;\ 0, \sigma_\alpha^2\right) \tag{15}$$

where ϕ is the normal density. The Z_i's are estimated as latents in the E-step, while α and σ_Z^2 are estimated as parameters in the M-step. Each, of course, need a new formula, including α whose formula in Equation (9) will have to be slightly adjusted.

5.3 Robustness to Low p and non-Lognormal Data

Initially, I hoped to be able to code up Section (3.2)'s proposed algorithm for this project. The lack of a closed form solution for α and β made it just a bit too hard to finish by the project deadline, but it's still absolutely doable with a bit more work that I hope to continue. With code written, simulations can be used to test how important each assumption is to the functionality of the model. This means I can set the scaling law parameters, select a bunch of N_i's and D_i's according to the lognormal distribution, and calculate the C_i's from them. Then, I can use the proposed EM algorithm to estimate the scaling law parameters back, and see how close they were to their "true" values (the values that I set them to to generate the data).

If model proposed in Section (3.1) is correct, then the parameters estimated with EM should have no problem converging to their true values, given enough C_i's and save the identifiability concerns addressed in Section (3.2) and nonconvexity concerns with EM in general. Of course, like any model, the one proposed in this project is not exactly true. But through this simulation method, we test how close it gets to estimating the true parameters anyway. For example, I have particularly cold feet about my decision to assume that N_i and D_i are lognormally distributed. So, I could try drawing them from a different distribution, calculating the C_i's with the true scaling law parameters, but then using the EM algorithm that assumed a lognormal

distribution to try to get the scaling law parameters back. If done enough times, this could even be used to estimate the bias of modeling decisions like the lognormal distribution in the face of specific different true models.

I'd also be particularly curious to use simulations to see how the model performs when the proportion of informed traders p is very low, as is probably the case in real life, especially if you don't try to estimate p in EM or your initial guess for it is way off. How does this affect the number of C_i's needed to accurately estimate the scaling laws, and how does this affect robustness to different kinds of model misspecifications?

5.4 Dynamics of Deceptive Labs

When AI labs purchase compute from a public market, they give away information about the scaling laws. As discussed in Section (4), sellers can use this information to set their prices. So, it's possible that labs may want to choose N and D that are not quite scaling-law-optimal in order to take information away from the seller.

In some cases, the buyer can erase information by adding noise to N and D from the optimal values. But in other cases, they may want their distortion to be directional. For example, Equation (11) suggested that under certain conditions, if the seller figures out that α and β are far apart, they'll have to lower their prices. So, if the buyers can convince the sellers of this, they may get cheaper compute. Of course, this will come at the cost of training suboptimal models, so whether this is a good idea depends on the buyer's utility function. Furthermore, if there are many buyers and deception requires cooperation between them to bias enough orders, this sets up a tragedy of the commons problem. And even worse, if the seller figures out that the buyer is directionally biasing their purchases, perhaps because they used the same logic as the buyer themselves, they can try to recover the unbiased scaling laws anyway. This leads to a reciprocal dynamic that's very difficult to work with due to the presence of nonlinearity, but would be fascinating to further study.

6 Works Cited

[1] Albert Kyle. Continuous Auctions and Insider Trading. Econometrica, 1985.

[2] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. arXiv:1712.00409, 2017.

[3] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. arXiv:2001.08361, 2020.

[4] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. arXiv:2203.15556, 2022.

[5] OpenAI. Learning to Reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, 2024.

[6] N. Sharghivand, F. Derakhshan, N. Siasi. A Comprehensive Survey on Auction Mechanism Design for Cloud/Edge Resource Management and Pricing. IEEE Access, 2021.

[7] https://vast.ai/. Accessed Dec 2024.

[8] https://sfcompute.com/. Accessed Dec 2024

[9] Greg Osuri and Adam Bozanich. AKT: Akash Network Token & Mining Economics. 2020.

[10] Paul Timofeev. The Case for Compute Depins. Shoal Research, 2024.

[11] Jean-Michel Glachant, Paul Joskow, Michael Pollitt. Handbook on Electricity Markets. Elgar, 2021.

[12] Aruthur Dempster, Nan Laird, Donald Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, 1977.