Pseudocode for the model

GitHub - uvafan/takeoff_modeling_v2

At initialization:

1. EC = effective compute of the latest model is initialized
2. Initialize a pool of N experiments E_1…E_N, each of the form (SD, SP, ED, EM, RT, C, EA, AEM) where
    a. Each of the following represents the time it takes (in weeks, for 2023-level):
        i. ED = Experiment design
        ii. SD = Software design
        iii. SP = Software programming
        iv. RT = Wall clock runtime with default monitoring
        v. EA = Experiment results analysis
    b. AEM = expected algorithmic efficiency multiplier
3. Currently N=10k, the other variables are drawn from lognormals for each experiment
4. There's a separate variable constant across experiments: EM = Experiment monitoring efficiency (2023-level = 1. Gets lowered)
5. Definitions:
    a. **Serial pre-work:** ED + SD + SP
    b. Serial post-work: EA
    c. The overall time taken to run the experiment = ED + SD + SP + RT * EM[1] + EA
6. We'll focus on N=10 research teams as a proxy for AI speedup across the whole company. We could later extend that to model many research teams with varying levels of taste.

At each timestep (when a research team has finished a serial task):

1. If an experiment has runtime done and needs to be analyzed:
    a. The post-work is done for that then the total algorithmic efficiency (AE) is multiplied by AEM, which increases EC (along with a physical compute increase)
    b. There is a translation of the EC increase into speedup multipliers for each of the experiment components (besides RT). This set by a manually specified schedule, with interpolation (in log-linear space for some, log-log for others)
2. Otherwise if all ongoing experiments are in the process of running, AND there are less than Max_E=3 experiments running per team:[2]
    a. An experiment is selected by the research team with taste T >= 1. Higher T means the researcher will tend to select better experiments

[1] The assumption that EM doesn't have to be parallelized is "conservative" in that there are likely serial bottlenecks from EM as well, which aren't modeled

[2] Max_E should probably increase over time but currently it doesn't.

        i. When choosing an experiment, a team with taste T chooses T experiments randomly, then does the best one (by AEM / serial time required[3]).
    b. The experiment is started, then the serial pre-work time for that experiment passes
        i. An experiment that is initiated during a timestep carries over to future timesteps if its overall time is greater than the length
    c. Due to physical compute increases, the EC moves up without any AE improvement. Then speedup multipliers are adjusted as above.

Other stuff:

1. Constants are fit to roughly have the median run look reasonably accurate in the no-automation case.
2. There is a list of TODOs at https://github.com/uvafan/takeoff_modeling_v2/blob/main/TODO.md. One potentially important one that Daniel and I have discussed since I last edited the TODO.md is to make teams have varying tastes.

[3] This would ideally incorporate non-serial time as well but it currently doesn't as doing the dumb thing of adding them all together hurts

Scratch

- [ ] Make research taste varying
- [ ] Somehow edit the formula to get performance to strip out “effective compute”?
- [ ] Edit code model?
- [ ] Fit to Epoch curves?

2 things to figure out:

1. performance metric(s)
2. framework representing compute and efficiency combining, and ideally a way to validate to some extent with past data

1. what we want to know is how varying amounts of research effort + compute will translate into performance
2. we can look in the past and see between various data points how much increases in both research effort and compute translate into performance
3. what might be helpful in particular is the discrete-ish nature of compute jumps (but these also accumulate the research effort
4. we should look at performance on downstream tasks rather than perplexity, maybe/probably? But maybe fitting on perplexity is also fine

Look into the q: how much performance can you get out of systems with the same amount of compute, over time? A slightly shittier proxy for algo progress?

Maybe epoch has basically already done this?

Then after that we need a metric:

1. we could do perplexity, but huge difficulty would be translating to ai r&d speedup. Also alg improvements like PTEs that don’t primarily affect perplexity seem v important. One possibility here is:
    a. More compute moves along the compute -> loss -> downstream perf mapping
    b. PT alg improvements mean compute -> loss mapping changes
    c. FT/Scaffolding alg improvements loss -> downstream perf (e.g. t-agi) changes
    d. Then downstream perf (e.g. t-agi) might be further mapped to improvements at various AI R&D tasks…
    e. One advantage of this is that we can fit the compute -> loss stuff relatively well to historical data, so less making stuff up at that stage, but OTOH there will be still be making stuff up in translating loss to downstream perf / t-AGI, and this will be completely missing PTEs by default
2. we could do the intuition pump of “equivalent of progress between x and y year”, and include benchmark differences to pump that intuition. That seems relatively reasonable to me

3. we could do crossing the human range, which seems okay to me except unclear what happens after you're better than the best human (maybe you become better than X copies of the best human?)
4. we could do t-agi and fit to made-up "historical data"
5. we could do a benchmark relevant to ai r&d, real or made up. But feels kind of like skipping a step, idk
6. {EHL, human-t} vs. {speedup, ELO vs. human, num humans}
    a. How would things even work with 2-dim perf?

Keep in mind that we want to translate into speedups of (ai, human) teams. For existing AE it should maybe be additive rather than multiplicative?

Once we have a metric and framework, how to model alg improvements?
For modeling alg improvements, would prob depend on the metric. But I'd guess we'd want to include the substitute/complement/obsoleted stuff rather than just having them stay (but maybe first do the simpler staying thing?). My guess is this might fit historical data better.

Types of alg improvements:
1. immediate performance boost
    a. On task that could already be done, or…
    b. Enables a new task
2. enabling scaling (is this actually a thing? maybe?)

What happens to alg improvements over time:
1. - complemented: another improvement combines with it greater than the two on their own
2. - somewhat subtituted
3. - fully obsolete/substituted: get completely replaced by something else, e.g. architecture change that isn't used anymore

Should we explicitly split into base/FT/scaffolding and model the differences as diagrammed on whiteboard? PT+FT+S should be more than each only added

Should I be including data (efficiency) somehow in this? ugh. I guess data and parameter efficiencies are just ways to break up any alg improvement so maybe fine to leave combined?

Hmm so what I really need is a way to represent how much marginal algo improvements help, and how much marginal compute improvements help, such that I get reasonable overall results? Could I do something dumb like decide these locally in some way, then roughly fit it to Epoch data?

Was playing around with compute -> perf curves:
https://www.desmos.com/calculator/cquqo1y52y

What should be at the top of the doc

[TODO: intro: motivation + summary + appropriate very strong caveats]

[TODO: Easy-to-understand high-level description of the methodology]

[TODO: takeaways from the modeling exercise]