CSCI 145 Problem 21: You Have $10M of Compute; Spend It

July 25, 2026

Submission Instructions

Please upload your work to Gradescope by **11:59pm on DATE**.

- Turn in a single PDF of *handwritten* solutions. Typed (including LaTeX) solutions will not be accepted.
- Problems are grouped into three-week **cycles**. Once per cycle, you will meet with the grader and present the solution to *one* problem from that cycle, selected at random when you arrive — so bring your work and be ready to explain any of them.

Problem 21: You Have $10M of Compute; Spend It

Congratulations: you run the training team at a new AI lab, and the board has wired you $10 million for one training run. At roughly $2 per GPU-hour, with each GPU sustaining about 6×10^{13} floating-point operations per second, that is 5 million GPU-hours $\approx 10^{24}$ FLOPs. Two decisions determine what the money buys: *which architecture* reads sequences more cheaply (Part A), and *how to split* the budget between model size and training data (Parts B and C). In 2022, the *Chinchilla* paper answered the second question with exactly the calculus below, and the answer reset how every major lab sizes its models. Throughout, n is the sequence length, d the model width, N the total parameter count, and D the number of training tokens, as in lecture.

Part A: The Cost of Reading a Sequence

Lecture counted attention's cost for a multi-head layer; here you redo the count in the simplest case and then compute the cost of the recurrent alternative the transformer displaced. Work with a single attention head of full width ($d_k = d$), so $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ are already computed. Count the multiplications in the score matrix $\mathbf{Q}\mathbf{K}^\top$ and in the value-weighted sum $\mathrm{softmax}\big(\mathbf{Q}\mathbf{K}^\top/\sqrt{d}\big)\mathbf{V}$, and count the entries the attention matrix occupies in memory, to conclude that attention costs $O(n^2 d)$ time and $O(n^2)$ memory.

The pre-transformer alternative was the **recurrent network**: read one token at a time, left to right, carrying a running summary $\mathbf{h}_i \in \mathbb{R}^d$ updated by $\mathbf{h}_i = \sigma(\mathbf{W}\mathbf{h}_{i-1} + \mathbf{U}\mathbf{x}_i)$ with $\mathbf{W}, \mathbf{U} \in \mathbb{R}^{d \times d}$. Show that processing the full sequence costs $O(nd^2)$ time.

Now form the ratio of the two time costs and conclude that attention is the more expensive mixer exactly when $n > d$. For a model of width $d = 4{,}096$: which architecture reads a 1,000-token email more cheaply, and how many times more does attention pay than recurrence on a 100,000-token codebase? Close with one caveat, in one sentence: the recurrence's n steps must run *in order* (step i needs $\mathbf{h}_{i-1}$), while attention's $n^2 d$ work is two big matrix multiplications, so which architecture actually finishes first on the parallel hardware that revived neural networks?

Part B: The Allocation Problem

Lecture gave the empirical form of the loss after training a transformer with N parameters on D tokens:

$$L(N, D) = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + L_\infty, \qquad A, B, \alpha, \beta > 0,$$

with L_∞ the irreducible floor (the Regression lecture's irreducible error, at industrial scale). Training costs about 6 FLOPs per parameter per token, so your budget fixes the *product*: $C = 6ND$. Absorb the constant by writing $\tilde{C} = ND$.

Using a Lagrange multiplier λ for the constraint $ND = \tilde{C}$ (or, equivalently, substituting $D = \tilde{C}/N$ and using one-variable calculus), show that the optimal allocation satisfies

$$\alpha \frac{A}{N^\alpha} = \beta \frac{B}{D^\beta},$$

then interpret the condition in one sentence: at the optimum, how does the loss reduction bought by a 1% increase in N compare with the one bought by a 1% increase in D, and what does that say about trading a little model for a little data? Argue the critical point is a true minimum by checking the constraint's two extremes: what happens to L as $N \to 0$, and as $N \to \infty$ with $D = \tilde{C}/N$?

Then combine the condition with $ND = \tilde{C}$ to derive the compute-optimal split

$$N^\star \propto \tilde{C}^{\,\beta/(\alpha+\beta)}, \qquad D^\star \propto \tilde{C}^{\,\alpha/(\alpha+\beta)},$$

and evaluate both exponents at the fitted values $\alpha = 0.34$, $\beta = 0.28$. When the budget grows $100\times$, roughly how much bigger should the model get, and how much more data should it see?

Part C: Spend the Money

Your Part B exponents came out nearly equal, so simplify with $\alpha = \beta$. Show that both exponents become exactly $\frac{1}{2}$, so the optimal N and D each grow like $\sqrt{\tilde{C}}$, and that the optimal ratio

$$\frac{D^\star}{N^\star} = \left(\frac{B}{A}\right)^{1/\alpha}$$

is a *constant*, independent of the budget. (Read it off the stationarity condition of Part B with $\beta = \alpha$, before you substitute anything.) The Chinchilla measurements put that constant near **20 tokens per parameter**.

Now spend: with $C = 6ND = 10^{24}$ FLOPs and $D = 20N$, solve for $N^\star$ and $D^\star$. Then audit the most famous model of the pre-Chinchilla era, GPT-3, which used $N = 175$ billion parameters and $D = 300$ billion tokens: how many tokens per parameter is that, and what N and D would your formula have prescribed for its budget $C = 6ND \approx 3.2 \times 10^{23}$? Conclude in one sentence whether GPT-3 was too big or too small for its budget. This audit is not hypothetical: Chinchilla (70B parameters, 1.4 trillion tokens, i.e., 20 tokens per parameter) outperformed models four times its size trained at the same cost.

Where this goes next. A one-page calculus exercise reallocated the industry's training budgets. The power-law form of $L(N, D)$ was measured, not derived, but once measured, the course's optimization toolbox (a Lagrange multiplier, a boundary check) turned $10 million of ambiguity into two numbers, and "Chinchilla-optimal" became the industry's sizing rule overnight. Part A is the course's complexity-accounting thread doing production work: the same $O(\cdot)$ bookkeeping that once compared normal equations against gradient descent now decides architectures. Attention's $O(n^2)$ cost is still an open problem; cheaper approximations of attention are an active research area precisely because of the count you did in Part A. Next lecture supplies the one ingredient today's block still lacks: a way for attention to know *where* its tokens are.