EECS 182 HW3 Participation A Report: Gemini 3 Pro on Non-coding Questions

[John Wang]

December 7, 2025

Executive Summary

In this report I evaluate the behavior of **Gemini 3 Pro** on the non-coding parts of HW3 (Problems 1, 3, 4, and 5). Gemini is instructed to provide step-by-step mathematical reasoning without generating any code. For each sub-question, I first request a one-shot solution and then, when needed, perform a small number of follow-up interactions (asking it to self-check, to incorporate missing constraints, or to expand specific steps).

Overall, Gemini is usually able to produce coherent derivations for calculus and probability questions, and it often gets the qualitative behavior (e.g., scaling with width) correct on the first try. However, on more subtle questions involving implementation details (e.g., counting rematerialization costs) it makes systematic mistakes and requires strong steering and clarification to converge to the correct answer. In several places, its first answer is confident but incorrect; only after being confronted with a hint or with an explicit discrepancy does it revise its reasoning.

I use these interactions to characterize: (1) how frequently Gemini can one-shot a question, (2) what kinds of misconceptions and hallucinations appear, and (3) which prompting strategies are effective at correcting them.

1 Methodology

1.1 Model and Interface

- **Model:** Gemini 3 Pro ("Thinking with 3 Pro").
- **Interface:** Web interface at `https://gemini.google.com`.
- **Mode:** Default settings, no additional tools or plug-ins.

1.2 Homework Scope

Gemini 3 Pro was only used on the *non-coding* parts of HW3: conceptual questions, derivations, and short-answer theory questions. All coding questions were solved without LLM assistance.

I applied the model to:

- Problem 1 (Maximal Update Parameterization and related reading),
- Problem 3 (Policy Gradient and Reparameterization),
- Problem 4 (Tensor Rematerialization: operation counting),
- Problem 5 (Tensor Rematerialization: memory loads and disk latency).

1.3 Interaction Protocol

For each non-coding sub-question, I follow this protocol:

1. Start from a fresh conversation window initialized with a global instruction prompt (Section 2).
2. Provide the problem statement (text or screenshot) and ask for a one-shot solution to a specific subpart.
3. If the one-shot answer is not fully correct, use up to 2–3 follow-up prompts to:
   - ask the model to re-check its derivation;
   - explicitly request that it use certain constraints or hints;
   - ask it to expand skipped algebraic steps.
4. For each interaction, label the outcome as *correct*, *partially correct*, or *incorrect* based on my own solution.

I keep an annotated log of the entire conversation, including my own comments on the model's behavior and the steering strategies used.

2 Global Instructions to Gemini

Input

> You are helping me with a deep learning homework. Your goal is to provide the solutions and step-by-step reasoning only for the non-coding parts of the problems.
>
> Requirements:
>
> Show intermediate mathematical reasoning and all derivation steps.
>
> Use clear math notation and concise justifications.
>
> If the problem has subparts (a), (b), (c), answer each clearly.
>
> If some information seems missing or ambiguous, explicitly say so instead of guessing.
>
> Do not write or suggest any code in any programming language.
>
> When you are unsure, state your uncertainty and explain what additional information you would need.

Gemini Output

> I understand the requirements. I am ready to assist with the mathematical derivations, theoretical reasoning, and conceptual analysis for your deep learning homework.
>
> Please provide the problem statement, including any specific equations, constraints, or diagrams mentioned. I will break down the solution step-by-step, ensuring all notation is clear and intermediate steps are shown, without generating any code.

Student Annotation

The purpose of this initial prompt is to constrain Gemini to: (1) operate only on non-coding questions, (2) show all intermediate derivation steps, (3) avoid hallucinating missing information, and (4) explicitly signal uncertainty. Gemini's response correctly restates these requirements and agrees to follow them, so I treat this as the global instruction state for the rest of the homework session.

3 Per-Problem Interaction Logs

For each problem below, I include representative interaction snippets and my annotations. The full raw chat log is available separately and can be cross-referenced with this report.

3.1 Problem 1: Maximal Update Parameterization

Setup

Problem context (informal). We consider a linear layer $W \in \mathbb{R}^{d_2 \times d_1}$ with i.i.d. Gaussian entries, acting on a random input $x \in \mathbb{R}^{d_1}$ with i.i.d. Gaussian entries, producing $y = Wx \in \mathbb{R}^{d_2}$. Parts (a)–(c) analyze the RMS norm of y and of a sign-SGD update; later parts involve reading a research paper on Maximal Update Parameterization (muP) and a spectral condition for feature learning.

The exact wording of the problem is omitted here and can be found in the HW3 handout.

Part (a): Expected Squared RMS Norm

User Input.

> Please read this question from the image and solve part (a).

Gemini Output (Attempt 1). Gemini derives the expected squared RMS norm of $y = Wx$ as follows:

- Define the RMS norm

$$\mathrm{RMS}(y) = \sqrt{\frac{1}{d_2}\sum_{j=1}^{d_2} y_j^2}, \quad \mathrm{RMS}(y)^2 = \frac{1}{d_2}\sum_{j=1}^{d_2} y_j^2.$$

- For a single output coordinate $y_j = \sum_{k=1}^{d_1} w_{jk}x_k$, it observes that $\mathbb{E}[y_j] = 0$ and computes

$$\mathbb{E}[y_j^2] = \sum_{k=1}^{d_1} \mathbb{E}[w_{jk}^2 x_k^2] = \sum_{k=1}^{d_1} 1 = d_1,$$

  using independence and the fact that both w_{jk} and x_k are standard normals.

- By linearity of expectation,

$$\mathbb{E}[\mathrm{RMS}(y)^2] = \frac{1}{d_2}\sum_{j=1}^{d_2} \mathbb{E}[y_j^2] = \frac{1}{d_2} \cdot d_2 d_1 = d_1.$$

Student Annotation.

- **One-shot correctness:** Algebraically correct and matches the intended answer $\mathbb{E}[\mathrm{RMS}(y)^2] = d_1$.
- **Behavior:** This is a clean one-shot derivation given only the screenshot and the global instructions.

Part (b): Sign-SGD Feature Update

User Input.

Please read this question from the image and solve.

Gemini Output (Attempt 1). Gemini analyzes the change in features under a sign-SGD update:

- It writes the update matrix as $\Delta W = \mathrm{sign}(g_i x_i^\top)$ and the resulting feature change as
$$y' = \Delta W x_i.$$
- For the j-th coordinate,
$$y'_j = \sum_{k=1}^{d_1} (\Delta W)_{jk}(x_i)_k = \sum_{k=1}^{d_1} \mathrm{sign}(g_j)\,\mathrm{sign}(x_k)x_k = \mathrm{sign}(g_j)\sum_{k=1}^{d_1} |x_k|.$$
- Let $Z = \sum_{k=1}^{d_1} |x_k|$. Then $y'_j = \mathrm{sign}(g_j)\,Z$ and hence
$$\mathrm{RMS}(y')^2 = \frac{1}{d_2}\sum_{j=1}^{d_2}(y'_j)^2 = Z^2.$$
- It expands Z^2 and computes
$$\mathbb{E}[Z^2] = d_1 \cdot 1 + d_1(d_1-1)\cdot\frac{2}{\pi} = d_1 + \frac{2}{\pi}(d_1^2 - d_1),$$
using $\mathbb{E}[x^2] = 1$ and $\mathbb{E}[|x|] = \sqrt{2/\pi}$ for $x \sim \mathcal{N}(0,1)$.
- Therefore $\mathbb{E}[\mathrm{RMS}(y')^2] = d_1 + \frac{2}{\pi}(d_1^2 - d_1) \approx \frac{2}{\pi}d_1^2$ for large d_1.
- To normalize the expected squared RMS norm to 1, it solves for c in $c^2\mathbb{E}[\mathrm{RMS}(y')^2] = 1$ and obtains
$$c = \frac{1}{\sqrt{d_1 + \frac{2}{\pi}(d_1^2 - d_1)}} \approx \frac{1}{d_1}\sqrt{\frac{\pi}{2}}.$$

Student Annotation.

- **One-shot behavior:** Gemini produced this reasonably complicated derivation in a single attempt.
- **Reasoning quality:** The qualitative scaling ($\Theta(d_1^2)$, independent of d_2) is correct, and the moment calculations use standard Gaussian facts.
- **Verification:** The precise constant factors require careful checking, but the main point—that the update's magnitude grows with d_1^2 and needs to be scaled down—is captured correctly.

Part (c): Dependence Only on d_1

User Input.

> You gave an update rule that depends on d_1 but not on d_2. Why is this the case?

Gemini Output (Attempt 1). Gemini explains the dependence qualitatively:

1. **Dependence on input dimension d_1.** It rewrites y'_j as a sum over d_1 terms and notes that the variance (and thus expected squared magnitude) of the sum grows with the number of independent inputs. Intuitively, each neuron accumulates d_1 input contributions, so the signal strength per neuron increases with d_1.

2. **Independence from output dimension d_2.** The RMS norm averages over outputs:

$$\mathrm{RMS}(y')^2 = \frac{1}{d_2}\sum_{j=1}^{d_2}(y'_j)^2.$$

   If all outputs have the same distribution, then the numerator scales like d_2 while the $1/d_2$ factor in front cancels this dependence. Adding more output neurons adds more terms but also increases the normalization factor, so the average per-neuron magnitude stays the same.

It concludes with a short summary: d_1 matters because the neurons activation is a sum of d_1 inputs; d_2 does not matter because RMS measures an average per output neuron rather than a total sum.

Student Annotation.

- **One-shot correctness:** This is the correct qualitative explanation of why the update rule scales with d_1 and not d_2.
- **Behavior:** Good high-level reasoning; no follow-up prompting was needed here.

Research Question: Maximal Update Parameterization and Spectral Condition

The homework also asks several reading-based questions about the paper *Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer* and an associated spectral condition for feature learning.

Part (a): Interpreting Figure 1. User Input. I provided a screenshot of Figure 1 (training loss vs. learning rate for different widths, under standard parameterization and under muP) and asked Gemini to answer the subquestion.

Gemini Output (Attempt 1 and Revision). Gemini's first answer described the plots somewhat vaguely. After I followed up with the actual figure and asked it to "rethink the question using the provided figure," it produced a clearer answer:

- The two side-by-side figures both plot training loss (y-axis) versus learning rate (x-axis).
- *Left panel*: standard parameterization. As width increases, the optimal learning rate moves and the shape of the valley changes, so hyperparameters do not transfer directly across widths.

- *Right panel*: muP parameterization. As width increases, curves nearly align and the optimal learning rate stays roughly constant, indicating width-invariant hyperparameters.

Student Annotation.

- The first attempt was too generic; prompting it to re-examine the concrete figure led to a much sharper and more accurate description.
- This illustrates that pointing the model to a specific visual cue ("Figure 1" rather than the paper in general) can significantly improve its answer.

Part (b): Interpreting Table 3. **User Input.** I showed Table 3 and asked for the corresponding scaling in standard parameterization (SP) for Adam learning rate on hidden weights.

Gemini Output (One Shot). Gemini locates the bottom row ("Adam LR") and the rightmost column ("Hidden weights (SP)") and reads off the value 1. It explains that in SP the Adam learning rate does not scale with width, whereas in muP it is scaled by $1/\text{fan}_{\text{in}}$.

Student Annotation. This was a simple lookup question, and Gemini handled it cleanly with no prompting.

Part (c)–(e): Spectral Condition for Feature Learning. For the remaining subparts, I provided a short excerpt about a "spectral condition for feature learning" (sometimes called "Condition 1") and a desideratum on the RMS norms of features and their updates.

Part (c): Desideratum 1 and RMS Norm.

Gemini explains that the desideratum requires both the features h_ℓ and their changes Δh_ℓ to have Euclidean norms on the order of $\Theta(\sqrt{n})$ (or equivalently RMS norms $\Theta(1)$) at all layers. This ensures:

- *Stability:* activations neither explode nor vanish as the network width grows.
- *Feature learning:* the update Δh_ℓ is of comparable scale to h_ℓ, so features actually change rather than being frozen.

It then rewrites the desideratum explicitly in terms of RMS norms: $\text{RMS}(h_\ell) \approx \Theta(1)$ and $\text{RMS}(\Delta h_\ell) \approx \Theta(1)$ across layers.

Part (d): Rewriting Condition 1 in RMS Form and Implication for Upper Bounds.

Gemini interprets "Condition 1" as a bound on the spectral norm of the weight matrices. It informally argues:

- Bounding the spectral norm of a layer implies that the RMS norm of the output cannot exceed a constant times the RMS norm of the input, giving an upper bound on signal amplification.
- Chaining these bounds over many layers yields an $\mathcal{O}(1)$ upper bound on the RMS norm of features, which matches the desideratum.

Part (e): Assumption for Lower Bounds.

For the lower bound, Gemini notes that one needs the weight matrices to be "well-conditioned"—non-degenerate with singular values bounded away from zero—and assumes random initialization plus generic input position. Under these assumptions, the network will not collapse the signal, giving the corresponding $\Omega(1)$ lower bound on feature norms.

Student Annotation (c–e).

- These answers are broadly consistent with the intended high-level story: spectral norm controls maximal amplification, and non-degeneracy prevents collapse.
- Gemini is noticeably more verbose on these conceptual parts; I did not need to provide follow-up prompts, but some of the phrasing is informal and I would compress it for the final write-up.

3.2 Problem 3: Policy Gradient and Reparameterization

Problem 3 concerns the score-function (REINFORCE) estimator and the reparameterization trick for Gaussian policies.

Part (a): Score Function Estimator for Gaussian

User Input.

> Read this question from the image and answer part (a).

Gemini Output (One Shot). Gemini writes the objective as

$$\mathcal{F}(\mu) = \mathbb{E}_{x \sim p_\mu(x)}[f(x)],$$

expresses it as an integral, and applies the log-derivative trick:

$$\nabla_\mu \mathcal{F}(\mu) = \mathbb{E}_{x \sim p_\mu(x)}\big[f(x) \nabla_\mu \log p_\mu(x)\big].$$

For the Gaussian density $p_\mu(x)$, it computes

$$\nabla_\mu \log p_\mu(x) = \frac{x - \mu}{\sigma^2},$$

so the score-function estimator is

$$\nabla_\mu \mathcal{F}(\mu) = \mathbb{E}_{x \sim p_\mu(x)}\left[f(x) \frac{x - \mu}{\sigma^2}\right].$$

Student Annotation. A standard derivation done correctly on the first try.

Part (b): Reparameterizing the Gaussian

Gemini Output. Given the same objective, Gemini rewrites the random variable x as

$$x = \mu + \sigma\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

and shows that this transformation preserves the distribution of x. Thus it expresses the objective as

$$\mathcal{F}(\mu) = \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)}\left[f(\mu + \sigma\varepsilon)\right].$$

Student Annotation. The reparameterization step is straightforward and was handled cleanly.

Part (c): Reparameterization Gradient

Gemini Output. Starting from the reparameterized form, Gemini moves the gradient inside the expectation and applies the chain rule:

$$\nabla_\mu \mathcal{F}(\mu) = \mathbb{E}_\varepsilon\big[\nabla_\mu f(\mu + \sigma\varepsilon)\big] = \mathbb{E}_\varepsilon\big[\nabla_x f(x)\big]_{x=\mu+\sigma\varepsilon}.$$

It concludes with the usual reparameterization gradient estimator:

$$\nabla_\mu \mathcal{F}(\mu) = \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\big[\nabla_x f(\mu + \sigma\varepsilon)\big].$$

Student Annotation. This is conceptually correct, though it glosses over the distinction between ∇_μ and ∇_x by identifying $\partial x/\partial\mu = I$.

Part (d): General Score Function Formula

Gemini Output. For a general parameterized distribution $p_\theta(x)$, Gemini repeats the log-derivative derivation and arrives at

$$\nabla_\theta \mathcal{F}(\theta) = \mathbb{E}_{x\sim p_\theta(x)}\big[f(x)\nabla_\theta \log p_\theta(x)\big],$$

explicitly connecting this general formula back to the Gaussian special case found in part (a).

Student Annotation. A textbook derivation, done correctly in one shot.

Part (e): Relating the Two Estimators

Gemini Output. Using the previous parts, Gemini sketches a proof that the reparameterization estimator corresponds to applying the chain rule to the same underlying expectation, while the score-function estimator corresponds to applying the log-derivative trick. It then identifies the Gaussian reparameterization of parts (b) and (c) as a special case of the more general formula.

Student Annotation. The argument is correct at the level of ideas, though somewhat verbose. I did not need to correct any mathematical errors here.

3.3 Problem 4: Tensor Rematerialization (Forward Ops)

Problem 4 asks for the number of forward operations required when using a specific tensor rematerialization scheme for a 10-layer network, given certain stored checkpoints.

Part (a): Counting Forward Operations

User Input.

> Based on the Tensor Rematerialization figure, compute the total number of forward operations needed during a single backward pass.

Gemini Output (Attempt 1: 28 operations). Gemini breaks the network into two segments based on checkpoints and counts, for each layer, how many forward passes are needed to recover its inputs from stored checkpoints:

- Segment 2 (layers 6–10) and Segment 1 (layers 1–5) are analyzed separately.
- For each layer it writes out a path such as "load A_5 and run forward through layers 6–9" and increments a counter by the number of forward passes on that path.
- It sums to 14 forward ops for layers 6–10 and 14 for layers 1–5, concluding with a total of 28.

User Feedback. I pointed out that the official solution gives 20, and that the discrepancy comes from counting some pre-computed activations twice (e.g., re-counting the forward passes required to produce a stored checkpoint).

Gemini Output (Attempt 2: 20 operations). After re-reading the problem, Gemini revises its logic:

- It distinguishes between (i) forward passes needed to *compute* a checkpoint the first time and (ii) re-using a stored checkpoint in later layers, which should not incur additional forward cost.
- With this corrected rule, it recomputes:
  - 10 forward ops for layers 6–10,
  - 10 forward ops for layers 1–5,

  for a total of 20.

Student Annotation.

- This is a good example where Gemini's initial reasoning is structurally plausible but misaligned with the exact cost model in the question. Only after I described the intended counting rule did it converge to the correct answer.
- For implementation-style questions, specifying the precise interpretation of "cost" is crucial; otherwise the model confidently overcounts.

3.4 Problem 5: Tensor Rematerialization (Memory Loads and Disk Latency)

Problem 5 refines the rematerialization analysis by counting memory load operations (denoted `loadmem`) and then compares it to a hypothetical scheme that loads activations from disk (`loaddisk`).

Part (b): Counting `loadmem` Operations

User Input.

> Using the Tensor Rematerialization scheme, count how many `loadmem` operations are used in one backward pass.

Gemini Output (Attempt 1: 12 loads). Initially Gemini treats both loading inputs from memory and re-using stored checkpoints as `loadmem` operations. It breaks the backward pass into two segments (layers 10 down to 6, and 5 down to 1) and counts a variety of loads, obtaining 12 total `loadmem` operations.

User Feedback. I clarified that the metric we care about is: for each layer, how many times we must *fetch* the necessary activation from memory in order to start the recomputation chain. In particular, if an activation is already cached due to a prior fetch within the same segment, it should not be counted again.

Gemini Output (Attempt 2: 10 loads). Gemini then simplifies the logic:

- For each of the 10 layers during backprop, we need exactly one "get the input tensor" operation to initiate recomputation from the nearest checkpoint.
- The output activations are assumed to be either stored or not part of the cost metric.
- This leads to a uniform pattern: 1 `loadmem` per layer, for a total of 10.

Student Annotation.

- The second attempt matches the intended answer. The main difficulty was aligning Gemini's mental model of the process with the exact definition of `loadmem`.
- This again illustrates that the model is sensitive to small ambiguities in how costs are defined.

Part (c): Disk Latency Comparison

User Input.

> Assume a new scheme that does not store activations in memory, but instead loads them from disk using `loaddisk` operations. Each backward pass still needs 10 such loads. Given that the total time of the tensor rematerialization scheme is 500 ns, find the latency per `loaddisk` so that the disk-based scheme matches this total time.

Gemini Output (One Shot). Gemini plugs in the numbers:

$$\text{TotalTime}_{\text{disk}} = (\#\text{loads}) \times x = 10x.$$

Setting $10x = 500\,\text{ns}$ gives $x = 50\,\text{ns}$ per `loaddisk` operation. It explicitly states:

> Each `loaddisk` operation should take 50 ns so that the total time equals 500 ns.

Student Annotation. A simple algebraic calculation; Gemini solves it correctly in one shot.

4 Aggregate Analysis and Reflections

Quantitative Summary

Across all subparts I asked Gemini to solve:

- **One-shot solutions (no correction needed):** Most analytical subparts (Problem 1(a,b,c), Problem 3(a–e), and Problem 5(c)) were solved correctly in a single attempt.
- **One-shot but requiring my verification:** For some derivations (e.g., the exact constants in Problem 1(b)), the structure was correct but I still needed to double-check details manually.
- **Multiple-attempt cases:**
  - Problem 1 research part (a): the first description of Figure 1 was vague; after I explicitly pointed Gemini to the figure, the second attempt was much sharper.
  - Problem 4(a): Gemini's first count of forward operations was 28; after I pointed out the intended cost model, it revised this to the correct answer 20.
  - Problem 5(b): similarly, the first count of `loadmem` operations was 12; with a better definition of what to count, it converged to 10.

Roughly speaking, Gemini could *one-shot* most purely mathematical derivations once the problem was clearly specified, but struggled with system-design-style counting questions where the cost semantics were subtle.

Common Error Modes

- **Ambiguous cost models.** When the question involved implementation details (rematerialization, memory loads), Gemini defaulted to a reasonable but non-matching interpretation and confidently used it until corrected.
- **Over-answering.** In several places it answered more than I asked (e.g., extending beyond a specific subpart), which is sometimes helpful but can also obscure what the question actually required.
- **Verbosity and informality.** On conceptual reading questions, the answers were often longer and less crisp than necessary, requiring me to distill the key points for my own understanding.

Effective Prompting Strategies

- Pointing Gemini to a specific figure/table and explicitly referencing the exact cell or curve greatly improved precision on reading-based questions.
- Asking it to "re-check" a derivation or to incorporate a missing condition was effective when the initial answer was close but incomplete.
- For implementation-style questions, I found it necessary to restate the cost definition in my own words before the model's reasoning aligned with the official solution.

Overall Reflection

Using Gemini 3 Pro as a collaborator on these non-coding homework problems was helpful for generating step-by-step derivations and for clarifying qualitative behavior (e.g., how norms scale with width). However, it was not safe to rely on it blindly: I still had to verify algebra, interpret ambiguous problem wording, and correct subtle miscounts.

From a participation-A perspective, this experiment shows that:

- For many analytic subparts, a modern LLM can indeed arrive at essentially correct solutions with relatively light prompting.
- For more implementation-flavored questions, it is easy for the model to confidently adopt the wrong interpretation unless the cost model is specified extremely precisely.