EE5102/CS6302 - Advanced Topics in Machine Learning – Fall 2025

Assignment 6 — *Release Date: 20 November 2025*

Instructions: Please read the following instructions carefully and abide by while preparing your submissions:

- This is the sixth graded assignment of the course which counts towards 7% of your final assignment aggregate.
- We need only one submission per group. If you submit the report for only your task and/or not in the required format, we will assign you zero marks.
- This assignment is due by **Thursday, 5 December 2025 on LMS**.
- As a submission, you need to prepare and submit you deliverables according to the instructions in **the last task of this assignment**.
- **AI Usage Policy:** You are not allowed to use any generative AI model—including LLMs—to write any part of your PDF report. The language/text and analysis must be entirely your own, in the report.

  For the coding part, you ***may*** use public libraries, built-in functions, or even get help from an LLM if needed. That's acceptable.

  However, once you submit your assignment, you are fully responsible for everything in it—text and code both. If your submission overlaps significantly with another group's work, you cannot later claim that some part was LLM-generated. That will not be accepted as an excuse.

Overview & Motivation

Reinforcement Learning from Human-Feedback

Reinforcement Learning from Human Feedback (**RLHF**) is the essential technique for aligning large language models (LLMs) with complex human values, ensuring outputs are helpful, harmless, and honest. Pre-training LLMs on vast datasets maximizes next-token likelihood, which often leads to outputs that are fluent but may be biased, toxic, or unhelpful. **Alignment** addresses this by introducing a human-defined reward signal, transforming the objective from simple data imitation to preference maximization. The standard RLHF pipeline involves collecting human preference data, training a separate **Reward Model (RM)** to predict these preferences, and finally fine-tuning the LLM policy (π) using this RM as the objective function. The choice of the final fine-tuning algorithm determines the complexity and stability of the alignment process.

Proximal Policy Optimization (PPO) PPO is the canonical and most widely used algorithm in RLHF, framing alignment as a standard reinforcement learning problem. After generating a model response, a Reward Model provides a scalar score, which may be either *sparse* (e.g., outcome-based) or *dense* (token-level or heuristic-based). PPO then computes the advantage for each sampled trajectory using a learned value function (critic), typically through temporal-difference or Monte Carlo returns:

$$A(x, y) = R(x, y) - V_\phi(x),$$

where $R(x, y)$ is the Reward Model's output and V_ϕ is the critic's estimate of expected return. The policy update uses a clipped likelihood-ratio objective to ensure stable training:

$$L^{PPO}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\min(r_\theta A,\ \text{clip}(r_\theta,\, 1-\epsilon,\, 1+\epsilon)\, A)\right] - \beta\, \text{KL}(\pi_\theta(\cdot|x)\,||\,\pi_{ref}(\cdot|x))\,,$$

where $r_\theta = \frac{\pi_\theta(y|x)}{\pi_{\theta_{\text{old}}}(y|x)}$. PPO is powerful but computationally heavy: it requires training both a Reward Model and a separate value function, making it more expensive, harder to tune, and more prone to instability due to critic prediction errors.

Direct Preference Optimization (DPO) DPO offers a significant simplification by eliminating the need for a separate Reward Model and the entire unstable RL sampling loop. DPO leverages the theoretical relationship between the optimal policy and the reward function (the Bradley-Terry model) to construct a direct classification loss based on collected human preference pairs (y_w, y_l). This approach converts the complex RL problem into a straightforward fine-tuning objective:

$$L^{DPO}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

DPO's primary advantages are its simplicity, stability, and computational efficiency. Its main limitation is that it directly optimizes the policy based on the explicit preference data, which may introduce biases such as a length bias.

Group Relative Policy Optimization (GRPO) is a reinforcement learning technique for fine-tuning large language models that removes the need for a separate value function (critic), unlike PPO. For each input prompt, the policy samples a group of responses (G). A sequence-level reward is computed for each response, which already includes a KL penalty against the reference model. GRPO then computes a *group-relative advantage* for each sample,

$$\hat{A}_i = \frac{R_i - \bar{R}}{\sigma_R},$$

using the group's mean and standard deviation as a lightweight baseline. This replaces the learned value function entirely. The policy is then updated using a PPO-style clipped objective applied to token log-probabilities. This approach provides strong stability and performance on reasoning tasks while being significantly more compute-efficient than PPO, since it removes the critic model and simplifies advantage estimation.

Mechanistic Interpretability

Deep neural networks are frequently characterized as "black boxes", we can train them to achieve superhuman performance on tasks ranging from protein folding to language generation, yet the specific algorithms they implement internally remain largely opaque. We know *that* they work, but we rarely understand *how*. **Mechanistic Interpretability** is a field that challenges this opacity. It seeks to reverse-engineer trained neural networks into human-understandable mechanisms, effectively "decompiling" the learned weights into legible algorithms. This approach stands in contrast to behavioral interpretability, which focuses on model outputs; instead, mechanistic interpretability looks inside the model to identify specific subgraphs, or **"circuits"**, that implement distinct behaviors. While the field is young, significant progress has been made in isolating these circuits.

- **Vision:** Cammarata et al. (2020) identified specific neurons responsible for detecting curves and shapes in image classification networks.
- **Language Models:** Elhage et al. (2021) and Olsson et al. (2022) developed the "Transformer Circuits" framework, discovering **Induction Heads**: circuits that allow models to copy patterns from the context, enabling in-context learning.
- **Specific Tasks:** Wang et al. (2022) completely reverse-engineered the circuit responsible for Indirect Object Identification (IOI) in GPT-2 Small.
- **Reinforcement Learning:** McGrath et al. (2021) found that AlphaZero explicitly acquires known human chess concepts, such as openings, during training.

In this assignment, we will use mechanistic interpretability to dissect a strange and counter-intuitive phenomenon known as **Grokking**.

First identified by Power et al. (2022), grokking describes a training dynamic where a model quickly achieves perfect accuracy on the training set (memorization) but remains at random chance on the test set for a prolonged period. Conventional wisdom suggests that if a model has overfit (zero training loss) and is not generalizing, training should be stopped to prevent further degradation. However, in grokking scenarios, if training continues for a massive number of additional steps, sometimes orders of magnitude longer, the test accuracy suddenly rises up to match the training accuracy.

This behavior raises fundamental questions: What is happening during that long, dormant plateau where test loss is flat? Is the model simply "stuck", or is it silently building a generalizing circuit that eventually overpowers the memorization circuit?

To investigate this, we will focus on the seminal paper *Progress Measures for Grokking via Mechanistic Interpretability*. You will train a small Transformer on the algorithmic task of Modular Addition:

$$a + b \equiv c \pmod{P}$$

where $P = 113$ is a prime number.

While this task appears simple, standard neural networks solve it in a highly non-intuitive way. Rather than memorizing the addition table (which it does initially), the network eventually invents a **Fourier Multiplication** algorithm. It utilizes trigonometric identities to perform addition via rotation in a hyperspherical embedding space.

A core component of this task involves analyzing the network's weights using the **Discrete Fourier Transform (DFT)**. While you may be familiar with DFT in the context of signal processing (converting a time-series signal into frequency components), its application here is a change of basis to reveal hidden structure.

In our task, the input tokens are integers $x \in \{0, 1, \ldots, P-1\}$. The Embedding Matrix W_E has shape $P \times d$, where d is the embedding dimension. This means every integer x is represented by a vector of size d.

- **Standard Basis:** If we look at the weights directly, they look like random noise. It is difficult to see any pattern in how the number "5" is related to the number "6".
- **Fourier Basis:** The mathematical operation of modular addition is intimately related to rotations around a circle (roots of unity). Therefore, we hypothesize that the model might be representing numbers as positions on a circle rather than independent categories.

To test this, we treat the vocabulary dimension P as a "time" domain and apply a 1D DFT to each of the d columns of the embedding matrix. This transforms our view from "what is the vector for token x?" to "what frequencies w_k does the model use to represent the sequence of integers?".

Why does this matter? The paper demonstrates that in the standard basis, the weights are dense and uninterpretable. However, in the Fourier basis, the weights become extremely **sparse**. The model essentially ignores most frequencies and concentrates all its "energy" on a few specific key frequencies. This sparsity proves the model isn't memorizing random numbers; it is constructing a precise algorithm based on sin and cos waves.

To perform this analysis, we will utilize **TransformerLens**, a library designed specifically for mechanistic interpretability of GPT-style language models. This library is used in the official implementation of the paper you are studying.

TransformerLens allows for easy "hooking" into the internal activations of a model (such as attention patterns, residual streams, and MLP outputs) without needing to modify the underlying source code. It treats the model as a computational graph where every intermediate state is accessible and modifiable.

You can access the library and its tutorials here. Familiarizing yourself with the concept of "hooks" in this library will be essential for the tasks involving circuit reverse-engineering and ablation.

Task 1: LLM Alignment. The objective of this assignment is to implement and analyze three distinct alignment methods: DPO, PPO, and GRPO. We will be using the SmolLM2-135M-SFT-Only as the SFT baseline model and the ORCA_DPO_Pairs instruction pair dataset. The core goal is to generate a detailed report that compares the trade-offs between how well the model adheres to preferences and the resulting degradation of the model's general capabilities. Furthermore, the analysis will focus on the stability of these methods and their known failure modes, such as Reward Hacking and Verbosity Bias. Due to computational constraints, you are encouraged to load the models in a 8-bit quantized format, and are free to use LoRA adaptors (PEFT) for any training/finetuning.

- You must first implement **DPO** on your smolLM model. Begin by preparing the dataset so that each example includes a prompt along with a preferred response y_W and a less-preferred response y_L in proper formatting. Once the data is organized, train the model for a few epochs using DPO. You can consider using tools such as the *trl* library, which provides a *DPOTrainer* that can handle the alignment setup directly.
- You must then implement **PPO** on your smallLM model. Start by training a reward model, where each example from the preference dataset is used to learn a scalar reward for preferred and less-preferred responses. This can be done by initializing a sequence-classification model with a single output label and fitting it on the chosen/rejected pairs. You may find the *AutoModelForSequenceClassification* from the *transformers* library helpful for this. After you have a trained reward model, you may proceed to the PPO phase. The policy generates responses to prompts, the frozen reward model assigns reward scores to those responses, the frozen reference model provides a baseline for KL-regularization, and the trainable value model estimates the expected return. At each iteration we compute advantages, update the policy while keeping it close to the reference model, and refine the value estimates. You should implement both a sparse reward setup (where only the final response receives a reward) and a dense reward setup (where token-level or intermediate rewards are used). You may be required to modify or extend parts of the *trl PPOTrainer* for dense and sparse reward setup. The *trl* library can still serve as a base for the PPO pipeline if you choose to use it, and LoRA adapters may be applied to the policy and value model to reduce computational cost.
- Finally, you must implement **GRPO** on your smallLM model. One possible approach is to begin by reusing the frozen reward model trained during PPO, since GRPO also relies on reward scores rather than a learned value function. Unlike PPO, GRPO removes the need for a separate critic and instead normalizes rewards within a group of sampled outputs. For each prompt, you may configure the training loop to sample a group of at least four candidate responses, compute reward scores for all of them using the frozen reward model, and then transform these rewards into normalized advantages based on their relative ranking within the group. The policy is then updated so that responses with above-group-average rewards are reinforced and those with lower rewards are suppressed. If you wish, you may use the *GRPOTrainer* provided in the *trl* library, though implementing the group-based normalization logic manually is also possible. LoRA adapters may again be applied to keep the computation feasible.
- We will now evaluate how well your aligned models perform and their potential side effects, such as catastrophic forgetting, verbosity bias, or reward hacking. First curate a small test set of 50 prompts, including open-ended questions to probe verbosity and "hack" prompts designed to trick the reward model (e.g., concise correct vs. long, plausible but incorrect responses). These prompts should be similar to the Preference Dataset or a held-out subset.
- To analyze **catastrophic forgetting**, you may track metrics such as KL divergence between the aligned policy and the frozen reference policy, which measures the drift from the original model, as well as perplexity on instruction-following data, which quantifies the alignment tax. Perplexity can be computed as

$$\text{Perplexity} = \exp\left(-\frac{1}{N} \sum_{t=1}^{N} \log P_\theta(y_t \mid x, y_{<t}) \right),$$

  where N is the total number of tokens in the dataset. Perplexity measures how well the model predicts the original SFT outputs: lower perplexity indicates that the model can still produce the original instruction-following responses accurately, while higher perplexity suggests that the model has "forgotten" some of its prior capabilities due to alignment fine-tuning. Comparing these metrics across DPO, PPO, and GRPO will help you understand which method preserves the original capabilities best.
- To evaluate **verbosity bias** in your aligned models, you may calculate mean, median, and standard deviation of response token counts across your test set, stratifying results by query type (factual questions should yield brief responses, while explanations allow moderate length). Critically examine the distribution shape where high variance with a right-skewed distribution may indicate occasional rambling episodes and potential length bias, while low variance suggests rigid length targeting regardless of task appropriateness. An adaptive model

should show variance that correlates with query complexity. Additionally, you may also prompt your models and explicitly instruct them to respond within specific limits (e.g., "in 50 words or less") and measure both compliance rate and deviation magnitude when limits are exceeded. Report your findings on which of your aligned models exhibit the most verbosity bias? Why may this be the case?

- To investigate **reward hacking**, begin by testing whether your reward model from PPO is overparameterized: take normal prompt–response pairs and create simple perturbations that preserve meaning but change the surface form (e.g adding filler phrases, reordering sentences, or injecting common "alignment" keywords). If these superficial edits significantly change the reward, your model is overfitting to shallow cues and is therefore vulnerable to exploitation. Next, design targeted hack prompts. These can be vague requests, safety-themed questions, or even impossible/contradictory tasks and compare how the PPO-aligned model responds relative to the base model. Use these targeted hack prompts on all 3 aligned models. Look specifically for cases where the PPO model gains a higher reward despite producing an incorrect, low-quality, templated answer. These divergences indicate reward hacking. You are encouraged to think creatively here: explore perturbations or prompt patterns you believe your reward model might be sensitive to, quantify reward shifts, and analyze whether PPO reliably exploits those weaknesses. The more systematic and curious your investigation, the better your understanding of reward hacking will be.

Task 2: Grokking and Reverse Engineering

1. **Setup:** You are required to establish the experimental environment and successfully reproduce the core phenomenon of grokking. You must create a dataset for Modular Addition using a prime modulus $P = 113$, formatted as sequences of `a`, `b`, `=` with target `c`. A critical aspect of this setup is the data split; you should use a training fraction of roughly 30% of the total 113×113 possible pairs. This specific scarcity of data is what forces the model to eventually generalize rather than relying solely on memorization. You will implement a 1-layer Decoder-only Transformer with an embedding dimension of $d = 128$, 4 attention heads, and an MLP hidden dimension of 512. Crucially, as noted in the paper, you must **not** use Layer Normalization, as it complicates the analysis without being necessary for convergence on this toy task. It is highly recommended to use the `HookedTransformer` from the Transformer Lens library. This configuration is represented in the `HookedTransformerConfig` as:

```
cfg = HookedTransformerConfig(
    n_layers = 1,
    n_heads = 4,
    d_model = 128,
    d_head = 32,
    d_mlp = 512,
    act_fn = "relu",
    normalization_type = None,
    d_vocab = p+1,
    d_vocab_out = p,
    n_ctx = 3,
    init_weights = True,
    device = device,
    seed = 999,
)
```

   A standard training loop is insufficient for observing grokking due to numerical instabilities. You must implement a custom Cross Entropy loss function rather than using PyTorch's built-in version. In your implementation, you must cast the logits to `float64` precision before computing the log-softmax and the final loss. This high-precision calculation is essential to prevent "slingshotting"—a phenomenon where numerical overflows cause massive spikes in loss that destabilize training. Furthermore, during model initialization, you should freeze all biases (set `requires_grad = False`) to ensure they remain zero throughout training. This simplifies the mathematical analysis in later parts by removing additive constants. Train the model using the AdamW optimizer with a learning rate of $\gamma = 10^{-3}$, momentum parameters $(\beta_1, \beta_2) = (0.9, 0.98)$, and a substantial weight decay of $\lambda = 1.0$ for approximately 10,000 to 40,000 epochs.

   Finally, you must generate a plot mirroring Figure 2 of the paper, logging Training Accuracy/Loss and Test Accuracy/Loss at every epoch. You should observe a distinct phase where Training Accuracy hits 100% while Test Accuracy remains near chance level, followed significantly later by a sudden jump in Test Accuracy. Analyze this plot and hypothesize why high weight decay is necessary for this transition, and why the optimizer eventually prefers the generalizing solution over the memorization solution.

2. **Reverse Engineering the "Fourier Circuit":** Now that you have a trained model that has successfully grokked the task, you will reverse-engineer its internal algorithm. The core hypothesis of the paper is that the model transforms inputs into a circular representation and performs addition via rotation. To verify this, you will first analyze the Embedding Matrix W_E $(P \times d)$. Perform a Discrete Fourier Transform (DFT) along the vocabulary dimension (P) for every embedding dimension. Visualize the l_2 norm of these Fourier components against the frequency k. You should observe that the matrix is sparse in the Fourier basis, identifying specific "key frequencies" where the model concentrates its energy.

   Next, you must investigate the mechanism by which the model combines these frequencies. The paper claims the model utilizes trigonometric identities, specifically $\cos(w(a+b)) = \cos(wa)\cos(wb) - \sin(wa)\sin(wb)$. Examine the activations of the Attention Heads and MLP neurons to verify this. Generate heatmaps of the attention patterns and MLP activations for your identified key frequencies. If the hypothesis holds, these heatmaps should exhibit clear, periodic, grid-like structures, indicating that the neurons are acting as quadratic terms that multiply sines and cosines.

   Finally, verify the "read-out" mechanism by analyzing the Unembedding matrix W_U (or the Neuron-Logit map W_L). The correct answer c should correspond to the value where cosine waves of different frequencies constructively interfere. For a specific query $a + b$, plot the logits contributed by the top key frequencies.

You must demonstrate that these logits align constructively (peak together) at the correct index $c = (a + b) \pmod P$ and destructively interfere at incorrect indices.

3. **Hidden Progress Measures:** Grokking appears to be a sudden, discontinuous jump in performance, but the central thesis of the paper is that the network is silently learning the solution in the background. Here, you will implement specific metrics to visualize this hidden progress. You will calculate two distinct loss curves: **Restricted Loss** and **Excluded Loss**. To compute Restricted Loss, project the model's logits onto *only* the key frequencies identified in Part 2, zeroing out all other frequency components, and evaluate this "clean" model on the test set. Conversely, for Excluded Loss, project the logits onto everything *except* the key frequencies and evaluate the loss on the training set.

   Plot these two curves over the entire course of training. Using these plots, you must segment the training history into the three phases described in the paper: **Memorization**, where the model uses random noise components to fit the training data (High Restricted Loss, Low Excluded Loss); **Circuit Formation**, where the Restricted Loss begins to drop, indicating the mechanism is being learned, while Excluded Loss rises as memorization fades; and **Cleanup**, the moment where Test Loss drops and the model sheds the noisy parameters. Provide a detailed analysis of the "Circuit Formation" phase, explaining how this proves that the model is amplifying a structured mechanism over time rather than learning suddenly.

4. **Ablation and Intervention:** To rigorously verify your mechanistic understanding, you must perform interventions on the model's weights. Take your fully trained model and manually "ablate" (zero out; this is called a zero-ablation) the Fourier components corresponding to the most dominant key frequency in the weights. Evaluate the model's accuracy after this excision. Does the performance drop to random chance, or does it only degrade partially? Perform the inverse experiment as well: ablate *all* frequencies except the key frequencies, effectively removing over 90% of the information in the weights. If the model still performs perfect addition, you have confirmed the "sparsity" hypothesis.

   Additionally, you will create a visualization to intuitively demonstrate the model's learned representation. Perform a Principal Component Analysis (PCA) on the embedding matrix W_E to reduce it to two dimensions. Create a scatter plot of the embeddings for tokens $0, 1, \ldots, P-1$ in this projected space. If the model has learned the rotational algorithm, the points should form a clear circle. Color code the points by their value modulo P to highlight the phase structure and confirm that the model is geometrically representing the modular arithmetic.

5. **Modular Subtraction (Non-Commutativity):** The standard Modular Addition task is commutative ($a + b = b + a$), which allows for symmetric processing. Now, you will test the robustness of the Fourier circuit hypothesis on Modular Subtraction ($a - b \pmod P$), where order matters and the operation is non-commutative. Train a new Transformer instance on the task $a - b = c \pmod P$ using the same hyperparameters and custom loss setup as before. Ensure the model successfully groks the task.

   Once trained, modify the trigonometric analysis from Section 4.2 of the paper to fit subtraction. The relevant identity changes to $\cos(w(a-b)) = \cos(wa)\cos(wb) + \sin(wa)\sin(wb)$. You must calculate the correlation between the attention head outputs and the terms of this specific trigonometric expansion. Analyze whether the network learns to invert the sign of the sine terms to handle the subtraction. Visualize the Fourier components of the embedding matrix and compare the sparsity pattern to the addition model to see if the non-commutativity forces a different frequency allocation.

6. **Geometry of the Loss Landscape:** The paper speculates that weight decay drives the model toward "simpler" solutions, but what does "simple" mean geometrically? Empirically analyze the curvature of the loss landscape. You are required to compute the top k (e.g., $k = 10$) eigenvalues of the Hessian matrix (the matrix of second derivatives of the loss w.r.t weights) at three specific checkpoints corresponding to the **Memorization**, **Circuit Formation**, and **Cleanup** phases identified in Part 3. Note that computing the full Hessian is computationally expensive, so you should use methods like Lanczos iteration or PyTorch's `hessian_eig` approximation.

   Analyze the magnitude of these eigenvalues across the three phases. During Memorization, does the Hessian indicate a "sharp" minimum (high eigenvalues), suggesting a brittle solution? Does the landscape significantly "flatten" (lower eigenvalues) during Circuit Formation and Cleanup? Correlate these findings with the "Gini coefficient" (sparsity metric) discussed in the paper. Discuss whether the increase in Fourier sparsity correlates with the flattening of the loss landscape, using this to explain why the "Cleanup" phase leads to a solution that is robust and generalizable.

7. **The Toy Model Debate:** Having successfully reverse-engineered a transformer, you might feel we have "solved" interpretability. Critically examine the extent to which these findings transfer to the broader field

of AI. Read Section 6 ("Larger models and realistic tasks") and Appendix F of the paper. Discuss why the clean "Fourier Circuit" discovered here is unlikely to appear in a Large Language Model (LLM) trained on internet text. Consider the nature of the task: Modular addition is strictly periodic and algorithmic, whereas natural language is not.

Explain why "circuits" in larger models (like GPT-4) might be harder to isolate than the single, clean circuit found in this assignment. Argue for the value of Toy Models. Why is it valuable to have a "ground truth" algorithm when testing interpretability tools? Finally, discuss the paper's suggestion that phase transitions might be inherent to learning "discrete" algorithms. Neural networks are continuous, yet algorithms are discrete; discuss how the "Circuit Formation" phase represents a struggle between these two modes.

Task 3: Synthesis of results & Report Writing Guidelines Now that you have conducted the experiments, the final task is to synthesize your findings into a coherent analysis and present it in a professional manner.

Deliverables:

- GitHub Repo: containing code (with documentation), any saved models or data subsets, and a README explaining how to run your analysis.
- PDF Report: ICML style, 6-10 pages including figures, addressing all points above. Treat it as a professional paper – clarity, structure, and correctness are key. Guidelines are given below. If you have additional results, you may add them in appendix after 10 pages. Write clearly and succinctly. Use bullet points or subheadings only if necessary.

Instructions:

- **Organize Your Code and Results:** Ensure your scripts for each task are clean, well-documented, and placed in the GitHub repo. Include instructions or scripts to install requirements and run the experiments. If some experiments are computationally heavy, provide saved outputs (e.g. learned model weights, logged metrics, or sample images) so the TAs/instructor can verify results without rerunning everything. The repository should be structured logically (perhaps a folder for Task1, Task2, etc., each with code and maybe a short README of its own).
- **Prepare Figures and Tables:** From Tasks 1–4, you likely have several plots, images, and metrics. Select the most meaningful ones to include in your report. DO NOT ADD JUST RANDOM FIGURES. Use clear captions and refer to them in the text. Ensure every figure is legible (use sufficient resolution as needed) and every table is properly labeled.
- **Writing the Report:** The report should roughly include:
  - **Abstract:** A short summary of what you did and key findings.
  - **Introduction:** Introduce the problem of DA/DG. State the objectives of this assignment that you will explore. Motivate why this study is important. You can briefly preview your approach and findings. Cite relevant background in proper academic citation format.
  - **Methodology/Experiments:** This can be structured by your tasks.
  - **Results:** Present the findings for each experiment, ideally intertwining the quantitative results with analysis. This is where you include those figures and tables.
  - **Discussion:** This is a crucial part to demonstrate deep reflection. Also discuss the interplay of architecture and data-driven biases and potential future designs.
  - **Conclusion:** A short paragraph wrapping up.
  - **Citations:** Throughout the report, if needed, cite sources in academic style. Make sure to cite any claim that is not your own result. A References section should list all cited works. Compare your findings with known literature to show deep understanding.
  - Finally, reflect on the process in a brief note (could be in the report discussion or a separate markdown in the repo): What surprised you? Did any results conflict with your expectations or published results?

References

[1] Curve Detectors Paper: Cammarata, Nick, et al. "Curve Detectors." Distill, 2020.
https://distill.pub/2020/circuits/curve-detectors

[2] Induction Heads Paper: Olsson, Catherine, et al. "In-context Learning and Induction Heads." arXiv, 2022.
https://arxiv.org/abs/2209.11895

[3] Transformer Circuits Paper: Elhage, Nelson, et al. "A Mathematical Framework for Transformer Circuits." Transformer Circuits Thread, 2021.
https://transformer-circuits.pub/2021/framework/index.html

[4] IOI Paper: Wang, Kevin, et al. "Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small." arXiv, 2022.
https://arxiv.org/abs/2211.00593

[5] AlphaZero Chess Paper: McGrath, Thomas, et al. "Acquisition of chess knowledge in AlphaZero." Proceedings of the National Academy of Sciences, 2022.
http://dx.doi.org/10.1073/pnas.2206625119

[6] Grokking Paper: Power, Alethea, et al. "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets." arXiv, 2022.
https://arxiv.org/abs/2201.02177

[7] Grokking Progress Paper: Nanda, Neel, et al. "Progress measures for grokking via mechanistic interpretability." arXiv, 2023.
https://arxiv.org/abs/2301.05217

[8] Superposition Paper: Elhage, Nelson, et al. "Toy Models of Superposition." arXiv, 2022.
https://arxiv.org/abs/2209.10652