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

Assignment 5 — *Release Date: 20 November 2025*

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

- This is the fifth 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 be investigating whether different models learn the *same* "reality"? To answer this, we must first address a fundamental barrier in interpreting Deep Learning models: **Polysemanticity**.

Early work in interpretability operated under the assumption that individual neurons were the fundamental unit of computation (e.g., a "dog neuron" or a "curve detector"). However, *Toy Models of Superposition* (Elhage et al., 2022) demonstrated that neural networks often represent more features than they have dimensions. To achieve this compression, models exploit high-dimensional geometry to store unrelated concepts in non-orthogonal linear combinations: a phenomenon known as **superposition**.

Because of superposition, inspecting a single neuron is often futile; it will activate for a confusing mixture of concepts (e.g., a specific neuron might fire for both "academic citations" and "yellow cars"). The meaningful units of computation are not the neurons themselves, but directions in the activation space.

To resolve this, researchers developed **Sparse Autoencoders** (Bricken et al., 2023; Cunningham et al., 2023). An SAE is a dictionary learning technique trained on the internal activations of a model. It attempts to decompose a

dense activation vector $x \in \mathbb{R}^d$ into a sparse linear combination of feature directions:

$$x \approx \sum_i c_i f_i$$

where f_i are unit vectors representing "monosemantic" concepts (concepts that mean one single thing), and $c_i \geq 0$ are sparse coefficients. By penalizing the L_1 norm of the hidden layer, SAEs force the representation to disentangle these superimposed features, effectively extracting the intelligible "variables" the model is using to think.

While SAEs allow us to interpret a single model, they do not inherently tell us if the learned features are arbitrary or fundamental. This brings us to the **Platonic Representation Hypothesis** (Huh et al., 2024).

This hypothesis posits that as deep learning models scale up and are trained on diverse data, their internal representations converge towards a shared statistical reality of the world: a "Platonic" ideal. If a ResNet and a Vision Transformer both achieve high accuracy on ImageNet, they must both understand the concept of a "wheel" or "fur texture," regardless of their wildly different architectures. Therefore, there should exist a mapping between their internal worlds.

You will verify this hypothesis empirically using **Universal Sparse Autoencoders (USAEs)**. By training a single, shared dictionary across multiple distinct source models, you force the autoencoder to find features that are not just useful for reconstruction, but are **universal** across architectures. If the Platonic Representation Hypothesis holds, the USAE should identify a shared latent space where Model A's representation of a concept aligns perfectly with Model B's representation. You will quantify this alignment, measure the "tax" paid to enforce it, and visualize the consensus between two distinct artificial intelligences.

Task 1: LLM Decoding Strategy Analysis The objective of this Task is to implement and conduct a comparative analysis of four fundamental LLM decoding strategies: **Greedy Search**, **Beam Search**, **Top-K Sampling**, and **Top-P Sampling** (Nucleus Sampling). You will use the SmolLM2-135M-SFT-Only as the SFT baseline model. The goal is to investigate the trade-offs between generation quality, output diversity, and the impact of the temperature hyperparameter. You may use any small, instruction-following dataset (e.g., a held-out subset of an instruction dataset) of your choice for the evaluation.

- First implement Greedy Search, which is the simplest decoding method, by selecting the token with the **highest probability** at each time step. To implement this, the model's logits for the next token must be computed, followed by a softmax operation to get the probability distribution over the vocabulary. The next token is then selected by finding the index corresponding to the maximum probability (the argmax operation). This process is repeated sequentially, appending the chosen token to the sequence until an End-of-Sequence (`[EOS]`) token is generated or the maximum length is reached.

- Next, we implement Beam Search, which improves upon greedy search by maintaining a fixed number, B (the **beam width**), of the most promising partial sequences (beams) at each step. To implement this, after generating the next token probabilities, all possible extensions (token additions) to the B current sequences are considered. The cumulative log-probability for each extended sequence is calculated as the sum of the log probabilities of its tokens. From this expanded set of sequences, only the top B sequences with the highest cumulative log-probabilities are kept for the next time step. This is repeated until the termination condition is met, and the sequence with the highest total score is selected as the final output. The key difference is that the search space is expanded by a factor of B at each step, significantly improving the chances of finding a globally more probable sequence compared to Greedy Search.

- Then we implement Top-K Sampling by introducing randomness by selecting the next token only from the **K** tokens with the highest probability mass. To implement this, the next-token logits are optionally scaled by a temperature T (where $T > 0$). After applying softmax to get the probabilities, the distribution is truncated, setting the probabilities of all tokens outside the top K to zero. The model then samples the next token *stochastically* from this reduced, re-normalized distribution.

- Finally, we implement Top-P Sampling, or **Nucleus Sampling**, which dynamically selects the smallest set of tokens whose cumulative probability exceeds a threshold **P**. Implementation starts by scaling the logits by temperature T. After softmax, the tokens are sorted by probability in descending order. The set of tokens, known as the **nucleus** ($\mathcal{V}_P$), is constructed by accumulating tokens until their cumulative probability exceeds the threshold P. The next token is sampled stochastically from the re-normalized distribution within $\mathcal{V}_P$. This is particularly effective because it restricts sampling to the reliably good tokens, allowing the size of the vocabulary set to adapt dynamically based on the sharpness of the probability distribution.

- For Top-K and Top-P sampling, run the model with a fixed K and P across a range of temperature values (e.g., $T \in 0.2, 0.5, 0.8, 1.0, 1.2$). We will investigate how both diversity and quality vary across temperature for all four sampling strategies by evaluating each generated sequence with two additional metrics. For diversity, we compute Distinct-N, where for each sample we count the number of unique unigrams, bigrams, or trigrams and divide by the total number of N-grams in that sample; higher Distinct-N values indicate richer vocabulary usage and lower repetition, making it a direct measure of lexical diversity that increases as temperature rises.

$$\text{Distinct-}N = \frac{\text{unique } N\text{-grams}}{\text{total } N\text{-grams}}$$

  Formally, this allows us to quantify how repetitive or varied the model's outputs become under different decoding regimes. For quality, score each generation using any pretrained open-source reward model of your choice, which provides a scalar estimate of correctness, coherence, and human preference. By generating multiple samples across a grid of temperature values and evaluating both metrics, we can ablate how increasing temperature affects the diversity–quality balance for both Top-K and Top-P sampling: diversity generally rises (higher Distinct-N) while quality tends to fall (lower reward scores), enabling us to characterize the tradeoff induced by each decoding strategy.

- To isolate the effect of decoding strategy on diversity at a fixed temperature (e.g., $T = 0.8$), we will use the **Distinct−N** metric (ratio of unique N-grams to total N-grams) in two complementary tests: Across-Prompt Diversity and Within-Prompt Diversity . For Across-Prompt Diversity, we generate one sample for each of N different prompts per strategy, aggregate all N results, and compute the resulting **Distinct−N** score to measure the global lexical variety over varied inputs. For Within-Prompt Diversity, we generate N samples for a single, fixed prompt per strategy, and compute the **Distinct−N** score on this small corpus to quantify the inter-sample variation produced when conditioned on the same input, thereby characterizing each strategy's tendency to collapse to similar responses.

Task 2: 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 3: Sparse Autoencoders In this task, you will implement a state-of-the-art interpretability technique: **Universal Sparse Autoencoders (USAEs)**. (official implementation available here). You will move beyond analyzing a single model to aligning the internal representations of different neural networks. You will critically evaluate the *Platonic Representation Hypothesis*: the idea that different models converge to the same fundamental concepts (features) regardless of architecture.

To facilitate your implementation, you are encouraged to utilize the **Overcomplete** library. This library provides optimized, GPU-accelerated implementations of various dictionary learning methods and SAE architectures (including TopK and JumpReLU). It will be particularly useful for this task as it is designed specifically for extracting concepts from large Vision models, abstracting away much of the boilerplate code required for efficient dictionary training.

You are required to train a USAE on the activations of two (or more) distinct pre-trained models (e.g., ResNet-18 and ViT-B trained on an image classification dataset of your choice). You can choose the type/number of models based on how much compute you have.

1. **Setup:** Let there be M models. Let $A^{(i)} \in \mathbb{R}^{d_i}$ be the activation vector from Model i. You must implement a shared feature space $Z \in \mathbb{R}^m$ (where $m \gg d_i$). The USAE consists of M encoders and M decoders, but they communicate via the shared bottleneck Z.

   - **Encoder $\Psi^{(i)}$:** Maps model specific activation $A^{(i)}$ to shared codes Z.
   - **Decoder $D^{(j)}$:** Maps shared codes Z to the reconstruction of Model j's activation $\hat{A}^{(j)}$.

   Implement the training procedure defined by the USAE framework:

   (a) **Select Source:** In each step, randomly select a source model i.

   (b) **Encode:** Compute the shared sparse code Z:

   $$Z = \text{TopK}(W_{enc}^{(i)}(A^{(i)} - b_{pre}^{(i)})) \quad \text{OR} \quad Z = \text{ReLU}(W_{enc}^{(i)}(A^{(i)} - b_{pre}^{(i)}) + b_{enc}^{(i)})$$

   Note: You may use TopK (k=32) as per the paper, or ReLU + L_1 penalty.

   (c) **Universal Decode:** Use this **same** Z to attempt to reconstruct the activations of **all** models $j \in \{1, \ldots, M\}$:

   $$\hat{A}^{(j)} = ZD^{(j)} + b_{dec}^{(j)}$$

   (d) **Optimization:** Minimize the aggregate reconstruction error:

   $$\mathcal{L} = \sum_{j=1}^{M} ||A^{(j)} - \hat{A}^{(j)}||_F^2 \quad (+\lambda||Z||_1 \text{ if using ReLU})$$

   Plot the training loss curves and a confusion matrix of R^2 reconstruction scores ($R_{i,j}^2$ (measures how well codes from Model i reconstruct Model j). Positive off-diagonal values confirm universality.

2. **Quantifying Universality:** Not all features are universal. Some are model-specific artifacts. You will implement the metrics used to distinguish them. Implement the Normalized Firing Entropy metric for each feature k in your dictionary:

   $$FE_k = -\frac{1}{\log M} \sum_{i=1}^{M} p_k^{(i)} \log p_k^{(i)}$$

   where $p_k^{(i)}$ is the proportion of activations for feature k that come from Model i.

   - $FE_k \approx 1.0$: The feature activates equally for all models (Universal).
   - $FE_k \approx 0.0$: The feature activates only for one model (Model-Specific).

   Plot a histogram of Firing Entropy across all learned features. Discuss whether you observe a bimodal distribution. What implications do your results have?

   Also, calculate the Co-Firing Proportion: For a given input image x, if Feature k fires for Model A, how often does it *also* fire for Model B? This measures instance-level alignment.

3. **Visualizing Consensus:** In standard interpretability, we look at dataset examples. In universal interpretability, we can perform **Coordinated Activation Maximization (CAM)**. Select 3 "High Entropy" (Universal) features. For each feature k:

(a) Initialize random noise images x_{start} for both Model A and Model B.

(b) Optimize x_A to maximize $Z_k(x_A)$ via Model A's encoder.

(c) Optimize x_B to maximize $Z_k(x_B)$ via Model B's encoder.

Display the resulting optimized images side-by-side. Do the two models see the same concept (e.g., both generate a spider web pattern)? Show one example where they diverge (the models agree the feature is active, but the visual representation differs).

4. **Analysis:** The paper argues that universal training is superior to training independent SAEs. Design an experiment (comparison with an independent SAE) to help verify this. Does forcing a feature to be shared across models degrade its quality? Check if the reconstruction score $R^2_{A,A}$ is lower in the USAE than in the Independent SAE. If R^2 drops, we are paying an "Alignment Tax." Discuss if the interpretability gain justifies this performance loss. Isolate features with **Low Firing Entropy**. Visualize them. The paper suggests unique features (like "depth" in DinoV2) are meaningful. In your experiment, are the unique features meaningful, or are they just noise/artifacts that the other model correctly learned to ignore?

   Did you observe "feature splitting"? (e.g., Model A has one "Dog" feature, Model B has "Dog Face" and "Dog Fur" features). How does the USAE bottleneck resolve this dimensionality mismatch? You used small models. The paper uses massive foundation models (SigLIP, ViT). Do you believe universality is an emergent property of *scale*, or is it fundamental to *learning*?

Task 4: 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

[9] Universal SAE Paper: Thasarathan, Harrish, et al. "Universal Sparse Autoencoders: Interpretable Cross-Model Concept Alignment." arXiv, 2025.
https://arxiv.org/abs/2502.03714

[10] Platonic Representation Paper: Huh, Minyoung, et al. "The Platonic Representation Hypothesis." arXiv, 2024.
https://arxiv.org/abs/2405.07987

[11] Monosemanticity Paper: Bricken, Trenton, et al. "Towards Monosemanticity: Decomposing Language Models With Dictionary Learning." Transformer Circuits Thread, 2023.
https://transformer-circuits.pub/2023/monosemantic-features/index.html

[12] SAE Features Paper: Cunningham, Hoagy, et al. "Sparse Autoencoders Find Highly Interpretable Features in Language Models." arXiv, 2023.
https://arxiv.org/abs/2309.08600