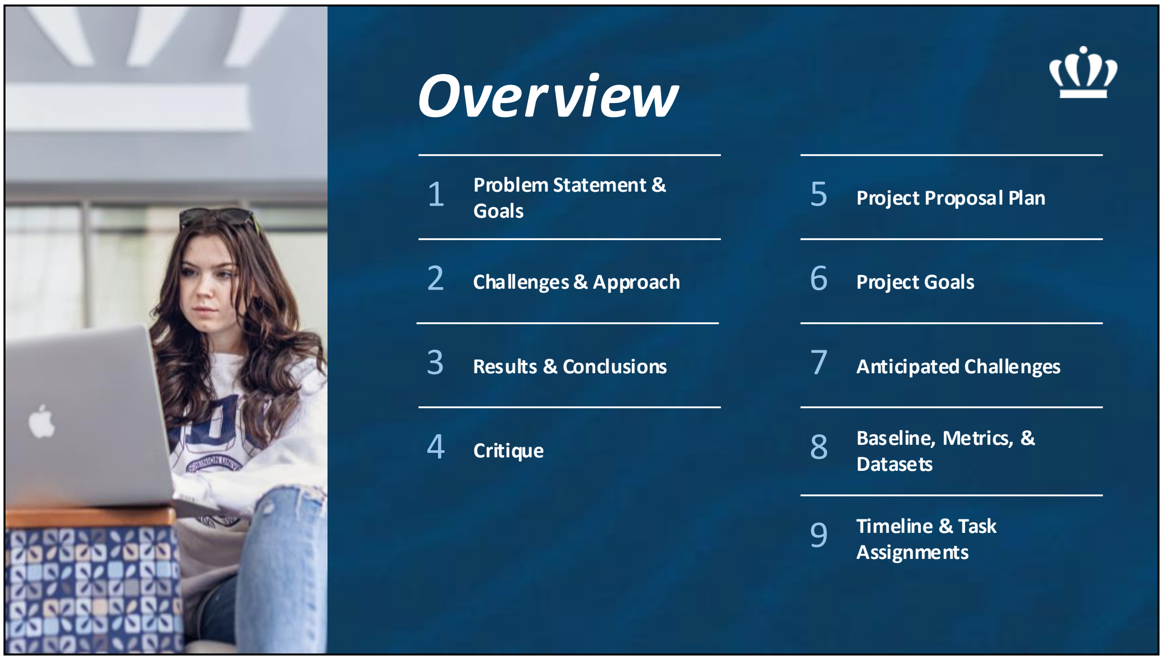
OLD DOMINION UNIVERSITY

Reproducing LoRA:

Low-Rank Adaptation of Large Language Models

Deshaun Jones, Brett Warren, Anthony Litwin
Date 10/21/2025

Speak to how we made the selection:
- Papers from top tier conferences tended to be use care agnostic
- Math looked familiar to concepts in class
- Material was not too dense to understand
- Topic appears recent enough and still relevant

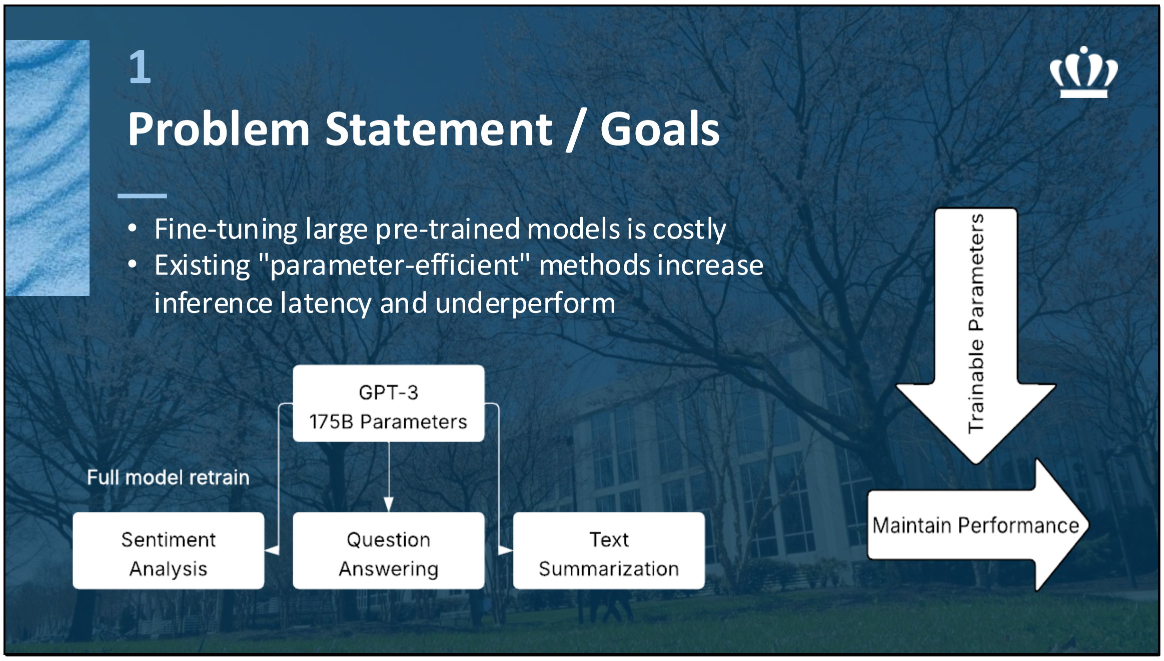


Overview

Good evening, everyone. We're the LowRankers, and today we'll be presenting our project on Reproducing LoRA: Low-Rank Adaptation of Large Language Models.

Here's a quick roadmap of our talk.
We'll begin with the problem statement and goals of the LoRA paper, then explain its challenges and approach.
Next, we'll summarize the main results and conclusions, followed by a critique highlighting the paper's strengths and limitations.
The second half of the talk covers our project proposal— what we plan to replicate, our goals, expected challenges, baselines and datasets, and finally, our timeline and task assignments.

Let's start with the motivation behind LoRA.



1

Problem Statement / Goals

- Fine-tuning large pre-trained models is costly
- Existing "parameter-efficient" methods increase inference latency and underperform

GPT-3
175B Parameters
Full model retrain
Sentiment Analysis
Question Answering
Text Summarization
Trainable Parameters
Maintain Performance

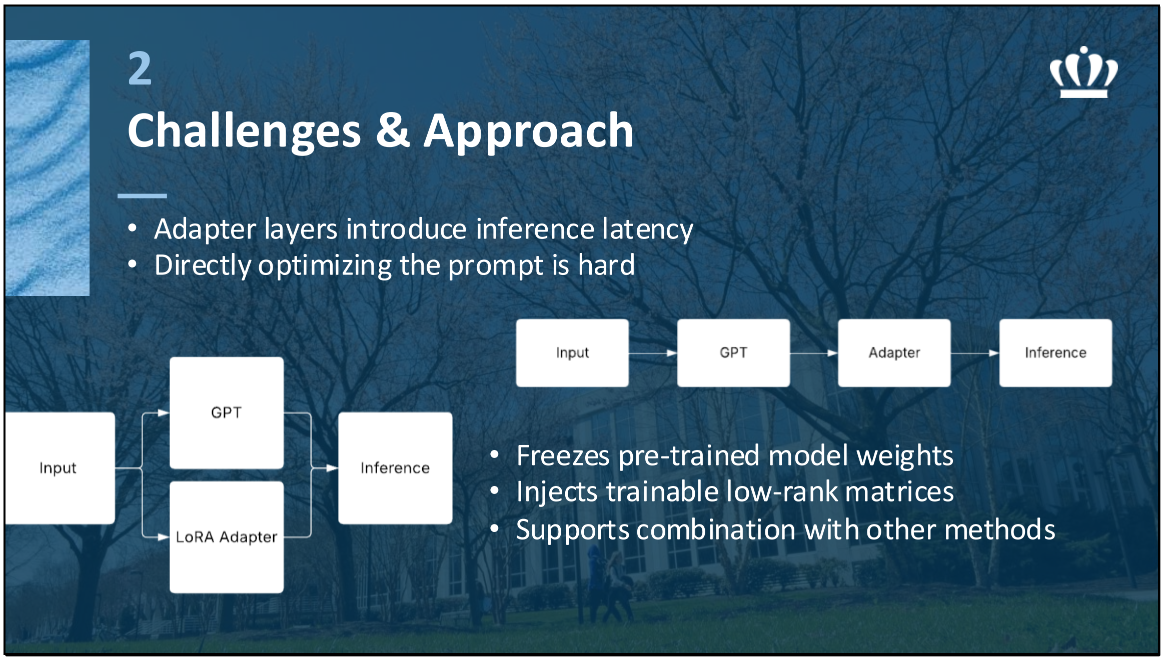
Problems:

- Modern NLP uses large pre-trained models which are then adapted for many downstream tasks.
    - Think pre-trained GPT model which are then fine-tuned to be a product expert, customer service rep, etc
- Full fine-tuning updates **all** parameters of the model for each task — so each new task means a full copy of the large model.
- With very large models (for example, ~175 billion parameters in GPT-3) this becomes impractical in terms of storage, deployment and compute.
- Existing parameter-efficient methods (e.g., adapter layers, prompt tuning) have trade-offs: they may introduce extra latency, reduce performance, or require non-trivial modifications.

Goals:

- Reduce trainable parameters and GPU memory requirements compared to full fine-tuning
- Maintaining or exceeding performance (accuracy) on benchmarks across different models
- Avoid inference latency increases or sequence length reductions seen in prior

- methods like adapters or prefix-tuning.
- Enable sharing of a single pre-trained model across tasks with small, swappable LoRA modules for better storage and deployment efficiency.



2 Challenges & Approach

- Adapter layers introduce inference latency
- Directly optimizing the prompt is hard

- Freezes pre-trained model weights
- Injects trainable low-rank matrices
- Supports combination with other methods

The authors thought there were two overarching challenges for this:

1. Adapter layers in existing solutions introduced inference latency
   - Adapter layers have to run sequentially after the main model for inference:
   - GPT -> Adapter -> Adapter -> Response
2. Directly optimizing the prompt is hard
   - Memory intensive
   - Costly during inference (sequence length + attention computation)
   - Difficult to optimize because its hard to learn

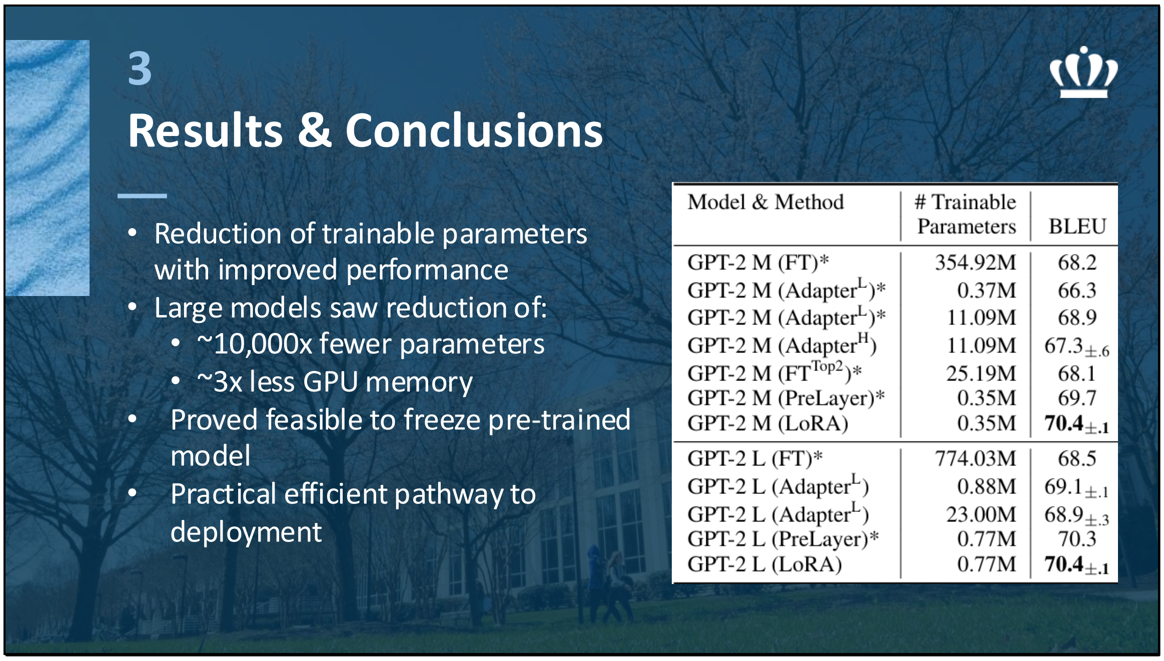
The approach they decided to take was to freeze the pre-trained model weights and attempt to inject changes that could be merged in parallel

An added benefit of this approach was that prefix tuning (or other approaches) could potentially be used in combination

Possible Question:
Intuition:

Pre-fix Tuning – Influencing the model via context hints
Technical definition – Add tokens to every input layer of the Transformer

Transformer layers generally targeted by LoRA: Multihead attention layer, Feed-Forward Layers
There are another several layers that are not altered



3

Results & Conclusions

- Reduction of trainable parameters with improved performance
- Large models saw reduction of:
  - ~10,000x fewer parameters
  - ~3x less GPU memory
- Proved feasible to freeze pre-trained model
- Practical efficient pathway to deployment

Model & Method	# Trainable Parameters	BLEU
GPT-2 M (FT)*	354.92M	68.2
GPT-2 M (Adapter^{L})*	0.37M	66.3
GPT-2 M (Adapter^{L})*	11.09M	68.9
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$
GPT-2 M (FT^{Top2})*	25.19M	68.1
GPT-2 M (PreLayer)*	0.35M	69.7
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$
GPT-2 L (FT)*	774.03M	68.5
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$
GPT-2 L (PreLayer)*	0.77M	70.3
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$

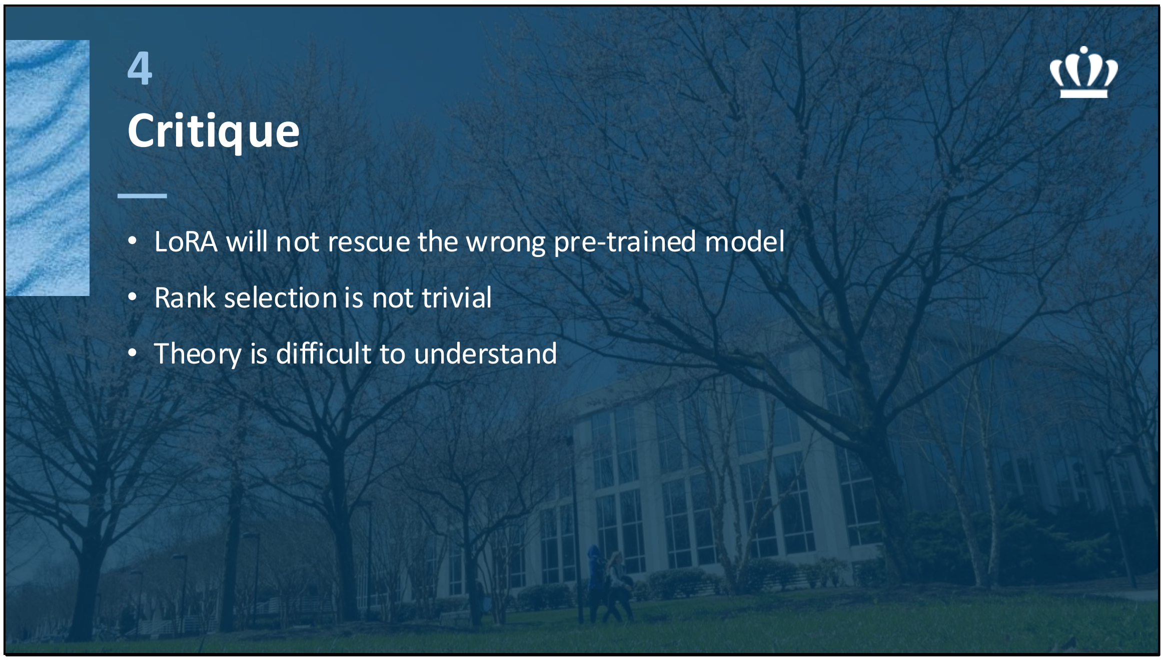
Results

- For the task of table-to-text generation using GPT-2 (Medium) (354 M parameters): full fine-tuning used ~354 M trainable parameters and achieved a BLEU score of 68.2. By using LoRA with only ~0.35 M trainable parameters, BLEU rose to 70.4 — i.e., better performance with ~1/1,000 the trainable parameters. arXiv+1
- Using the 175 B parameter GPT-3 as a reference, LoRA reportedly **reduced the number of trainable parameters by up to ~10,000×** and the GPU memory requirement by ~3× compared to full fine-tuning. arXiv+1
- In experiments varying the "rank" (r) of the low-rank update in LoRA, even very small ranks (e.g., r = 1 or 2) achieved nearly the same validation accuracy as larger ranks (e.g., r = 64) on tasks like WikiSQL and MultiNLI — supporting the claim that adaptation updates have low intrinsic dimensionality. arXiv

Conclusion

- The data show that you can freeze most of a large pretrained model and only train a tiny "adapter" (few trainable parameters) and still *match or exceed* full fine-tuning performance.

- With such dramatic reductions in trainable-parameter count and memory demand (for example, 0.1% or less of parameters in some cases), LoRA enables many specialised versions of a base model without retraining the full model each time.
- Because LoRA's updates can be merged into the frozen weights at inference, there is no inference-speed penalty despite the efficiency gains.
- The fact that very low ranks (r) suffice for adaptation implies the "task-specific change" needed is confined to a low-dimensional subspace — which means large models may be easier to adapt than one might assume.
- Overall, LoRA offers a practical, efficient pathway for adapting very large language models — making them more deployable and more economical in real-world settings.



4 Critique

- LoRA will not rescue the wrong pre-trained model
- Rank selection is not trivial
- Theory is difficult to understand

Here's a structured critique of the paper **Low-Rank Adaptation (LoRA)** (Hu et al., 2022) — a popular parameter-efficient fine-tuning method for large pretrained models. I'll cover what it does well, where it has limitations (both acknowledged and emerging), and open questions. If you like, I can also pull together community commentary and follow-up work.

✅ What LoRA does well

- **Parameter efficiency**:

LoRA re-parameterizes weight updates via a low-rank decomposition: for a frozen pre-trained weight matrix (W_0), it defines
[
W = W_0 + \Delta W,\quad \Delta W = B A
]
where (B \in \mathbb R^{m \times r}, A \in \mathbb R^{r \times n}) with (r \ll \min(m,n)). (Curriculum Vitae)
This reduces the number of trainable parameters substantially.

- **Minimal inference overhead**:

Because the main weights (W_0) are frozen and only a small "adapter" ((BA)) is trained, inference speed and memory footprint remain nearly the same as the original model. (Curriculum Vitae)

- **Good practicality / deployability**:
It has been adopted by many practitioners for fine-tuning large language models (LLMs) and vision models, especially when full fine-tuning is too expensive (in terms of GPU memory, storage, or deployment cost). The method's simplicity (just plug in low-rank adapters) makes it easy to use.
- **Preservation of base model knowledge / less catastrophic forgetting**:
Some studies find that because LoRA only adds small perturbations, the base model's general capabilities are less disrupted compared to full fine-tuning. For example, the work "LoRA Learns Less and Forgets Less" found LoRA better preserves out-of-domain tasks. (arXiv)

⚠️ Key limitations and critiques

Here are the major weak spots, caveats and areas of concern — some of which are discussed in the literature, and others that are emerging.

- **Expressive capacity / performance gap**
    - Because the adaptation is constrained to a low-rank space, there is inherently less flexibility compared to full fine-tuning. For certain tasks this means LoRA underperforms full fine-tuning. For example, the "Learns Less and Forgets Less" paper found that "In the standard low-rank settings, LoRA substantially underperforms full fine-tuning." (arXiv)
    - The recent theoretical work "The Expressive Power of Low-Rank Adaptation" shows that to approximate an arbitrary target model you may need LoRA rank to grow large (e.g., roughly width × (depth_target/depth_base)). (OpenReview)
    - Thus, the low-rank constraint imposes a trade-off: you save parameters, but you may lose capability unless you push the rank higher (which reduces the parameter savings).
- **Rank selection is non-trivial**
    - Choosing the rank (r) is a hyper-parameter. Too small → underfitting target task; too large → loses parameter-efficiency benefits.
    - Some survey work notes: "The research could benefit from more extensive ablation studies and theoretical analysis of rank selection criteria." (AIModels)
    - In practice, determining the "right" adapter rank for different layers/tasks remains somewhat ad-hoc.
- **Theoretical understanding**
    - While LoRA has been very successful empirically, the theoretical foundations (when/why it works, how to pick rank, how optimization proceeds) have lagged behind.

  - Recent works (e.g., "On the Optimization Landscape of Low Rank Adaptation Methods…" ICLR 2025) analyze the optimization landscape and show that LoRA may have more spurious local minima, or slower convergence, relative to full fine-tuning or alternative methods. (Proceedings ICLR)
  - Also, there are complexity-theoretic limits: "Computational Limits of LoRA Fine-Tuning for Transformer Models" shows phase-transition behaviour in algorithmic efficiency of LoRA updates. (Maojiang Su)
- **Task-specific and domain-specific limitations**
  - On very challenging tasks (e.g., code generation, math reasoning) the limited adapter capacity may be more of a bottleneck. For example, the RaSA paper points out that the "limited expressive capacity of LoRA … has been recognized as a bottleneck, particularly in rigorous tasks like code generation and mathematical reasoning." (arXiv)
  - Also, for vision tasks, concerns around fairness, subgroup performance, calibration, etc., have been raised: the "On Fairness of Low-Rank Adaptation of Vision Models" study shows that while LoRA doesn't always worsen fairness, "isolated examples do exist where LoRA exacerbates model bias across subgroups" in certain settings. (CS231n)
- **Frozen base weights may limit adaptation**
  - Because the base weights are frozen in LoRA, some kinds of adaptation that require modifying the original weights may be less effective. There might be cases where full fine-tuning of the base weights yields qualitatively different representations.
  - Additionally, the assumption that the fine-tuning update lies in a low-rank subspace may break down for some tasks or architectures.
- **Storage/management overhead of multiple adapters**
  - While each adapter is small, if you maintain many different adapters (for different tasks, domains, or versions) you still incur management overhead. Also, you might need to load/unload adapters dynamically.
  - Some practical workflows treat each adapter as a "module" to be swapped in/out, but that adds logistical complexity (and versioning, compatibility concerns, etc).

🔍 Emerging areas of critique / open questions

Beyond the well-known issues, the literature flags some open directions and subtle challenges:

- **Stability and generalization across domains**: Does LoRA always generalize gracefully to very different distributions than the one seen in fine-tuning? Some studies suggest that because it adapts less (fewer parameters), it may

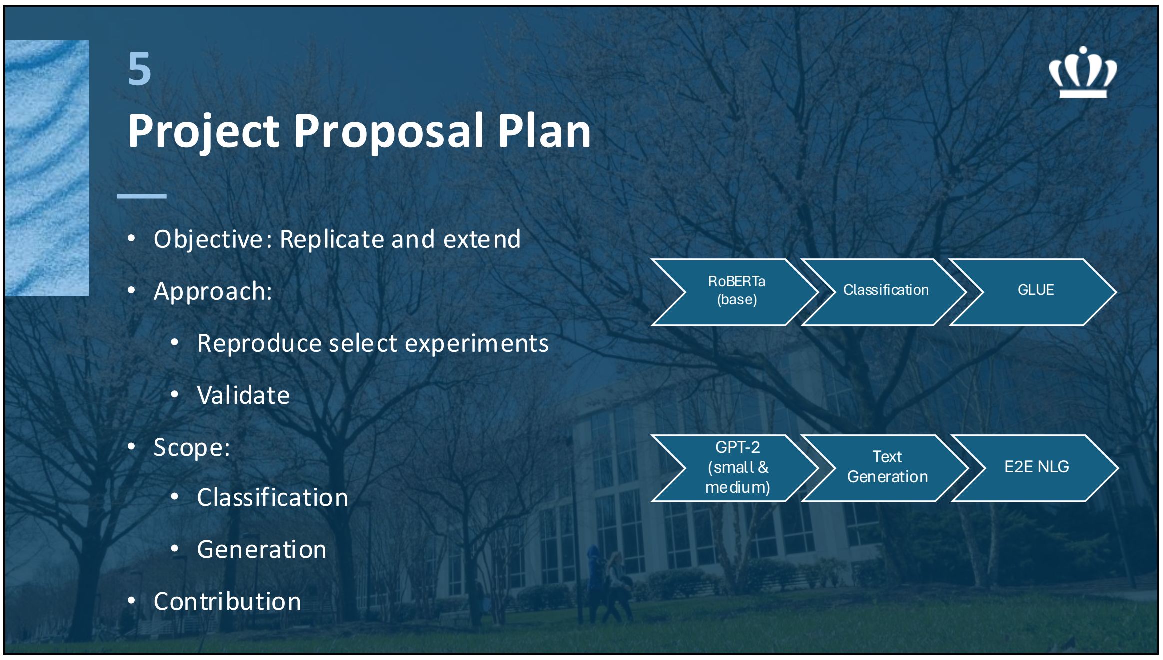
- generalize better, but there is less evidence in very different domain shifts.
- **Optimal rank allocation per layer/module**: Many implementations pick a single rank (or a small variety) across layers, but different layers may benefit from different adapter ranks. Some newer works target dynamic or adaptive rank assignment (e.g., GoRA, dynamic LoRA variants) to address this. (arXiv)
- **Robustness / adversarial behaviour / fairness**: The fairness study mentioned earlier shows that while LoRA in many cases behaves similarly to full fine-tuning, there are settings where subgroup inequalities increase. (CS231n)
- **Composability and multi-task / continual learning scenarios**: When you stack multiple adapters (for multiple tasks), do they interfere? Is there "adapter drift" or do earlier adapters interfere with later ones? The literature is still exploring how LoRA works in continual learning or multi-task settings.
- **Inference-time constraints vs adaptation time**: LoRA is great for inference efficiency, but some work shows convergence or adaptation takes longer than full fine-tuning, or performance plateaus earlier unless rank is increased.
- **Optimal initialization of adapter matrices (A, B)**: Some newer work (e.g., LoRA-GA) suggests that how you initialize the low-rank matrices matters a lot for convergence and performance. (arXiv)

🎯 Summary: When to use LoRA and when to be cautious

Use LoRA if:
- You have a large pretrained model, and fine-tuning all weights is too expensive (compute, memory, storage).
- You want to deploy multiple versions (e.g., domain-adapters) without duplicating the whole model.
- You're working in a domain where you expect the adaptation to be moderate, and you care about inference efficiency and modularity.

Be cautious / consider alternatives if:
- The downstream task is very different from pre-training domain or extremely challenging (e.g., high-complexity reasoning, adversarial robustness, major architectural shift). In these cases, full fine-tuning (or richer adapters) might perform better.
- You don't know how to choose the rank or how to allocate it across layers; hyper-tuning may become heavy enough to offset the savings.
- You care deeply about fairness, subgroup performance, or worst-case robustness and want to be sure no group's performance is harmed.
- You anticipate needing many adapters for many tasks and are concerned about management, versioning, interactions among adapters.

If you like, I can **dig into specific empirical studies** that compare LoRA vs full fine-tuning (with numbers) and **collect critiques/benchmarks** showing where LoRA still lags behind — would you like me to do that?



5

Project Proposal Plan

- Objective: Replicate and extend
- Approach:
  - Reproduce select experiments
  - Validate
- Scope:
  - Classification
  - Generation
- Contribution

RoBERTa (base)
Classification
GLUE
GPT-2 (small & medium)
Text Generation
E2E NLG

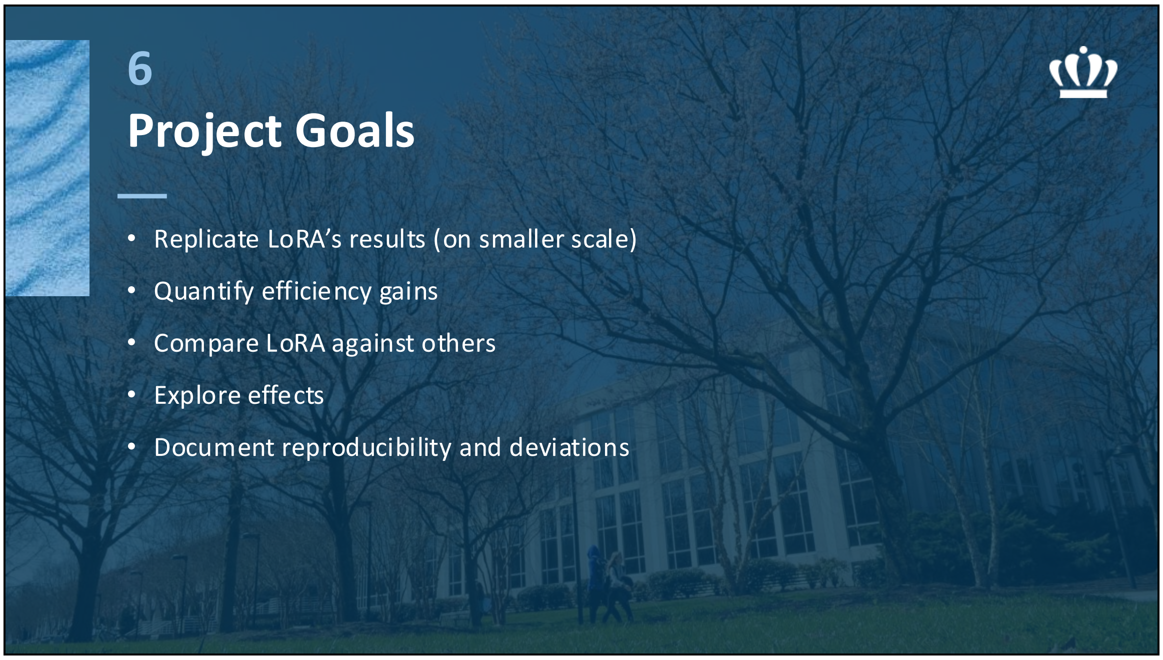
Building on the LoRA paper, our project focuses on replicating and validating its key findings, using models and datasets that are computationally practical for us.

Specifically, we'll reimplement LoRA on RoBERTa-base for classification tasks from the GLUE benchmark, and on GPT-2 small and medium for text generation using the E2E NLG Challenge dataset.

The goal is to confirm LoRA's reported efficiency and performance trade-offs and to test how they scale down to smaller models.

Along the way, we'll run ablation studies—changing the LoRA rank and the specific layers adapted—to see how these design choices affect accuracy and efficiency.

In short, our project aims to reproduce the core contributions of LoRA in a smaller, more controlled setting while analyzing what aspects matter most for its success.



6

Project Goals

- Replicate LoRA's results (on smaller scale)
- Quantify efficiency gains
- Compare LoRA against others
- Explore effects
- Document reproducibility and deviations

Our project goals are structured around both reproduction and analysis.

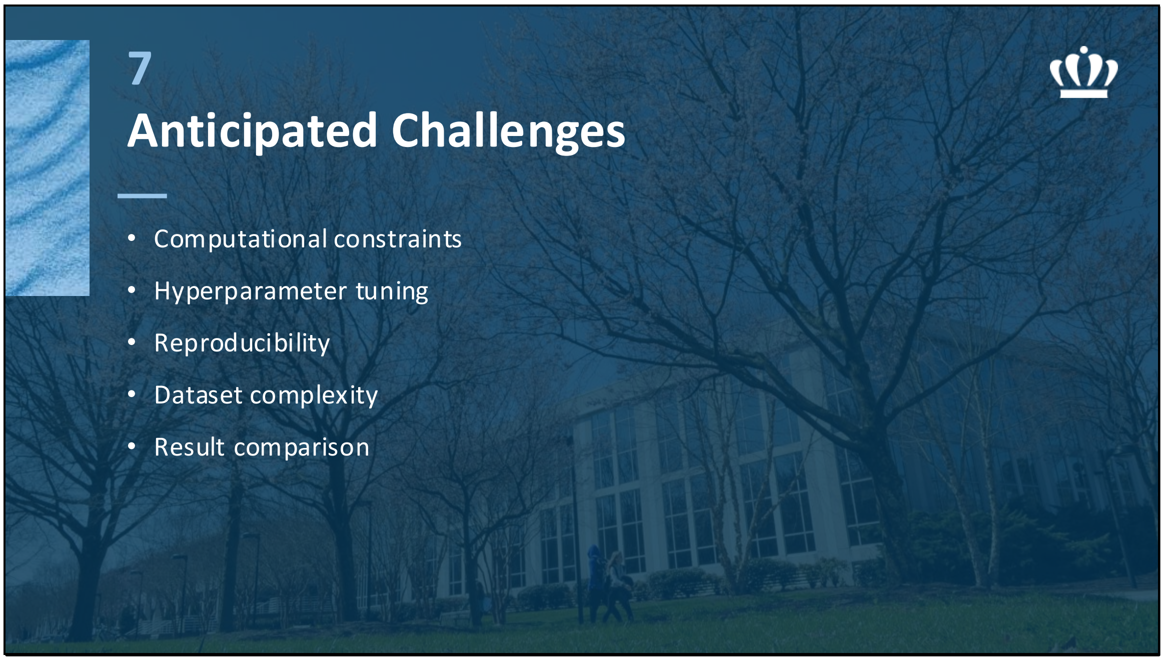
First, we aim to replicate LoRA's main results on smaller, open-source models. This means confirming that we can achieve similar performance while training only a small fraction of the model's parameters.

Second, we'll quantify efficiency improvements — how much we save in memory, computation, and training time compared to full fine-tuning.

Third, we'll compare LoRA to multiple baselines, including full fine-tuning, partial fine-tuning, adapters, and bias-only tuning.

Finally, we'll conduct ablation studies, varying the LoRA rank and the choice of adapted layers to understand what drives its effectiveness.

Together, these goals will help us evaluate not just whether LoRA works, but why and how well it works under constrained conditions.



7

Anticipated Challenges

- Computational constraints
- Hyperparameter tuning
- Reproducibility
- Dataset complexity
- Result comparison

While our project is manageable in scope, there are a few key challenges we anticipate.

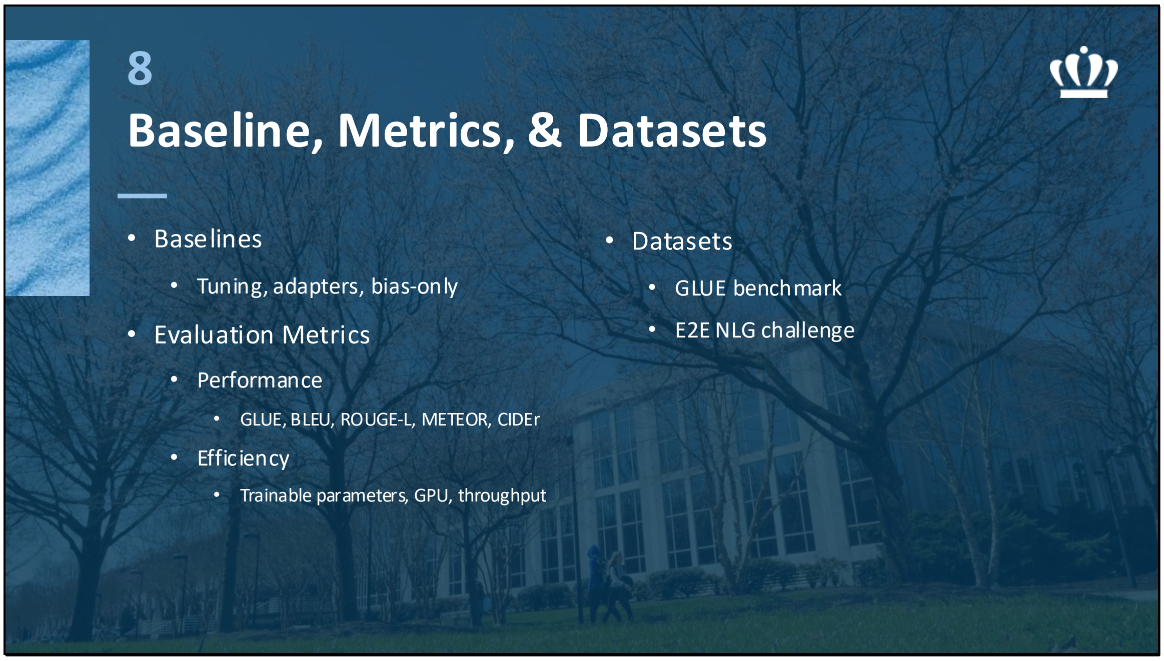
First, we're operating under limited computational resources, so we need to make careful choices about model size, batch size, and training epochs to stay within GPU limits.

Second, LoRA's performance can be sensitive to hyperparameters—especially the rank of the low-rank matrices and the learning rate—so tuning these efficiently will be important.

Third, ensuring reproducibility can be tricky. The original paper used large-scale hardware and specific environments, so our smaller-scale setup might introduce variability.

We'll also need to balance dataset complexity: selecting tasks that are diverse enough to be meaningful but small enough to train within our constraints.

Finally, comparing our results directly to the original paper's large-scale benchmarks will require some care—we'll focus on relative trends rather than absolute numbers.


8

Baseline, Metrics, & Datasets

- Baselines
  - Tuning, adapters, bias-only
- Evaluation Metrics
  - Performance
    - GLUE, BLEU, ROUGE-L, METEOR, CIDEr
  - Efficiency
    - Trainable parameters, GPU, throughput
- Datasets
  - GLUE benchmark
  - E2E NLG challenge

To evaluate LoRA’s effectiveness, we’ll compare it against several baseline methods.

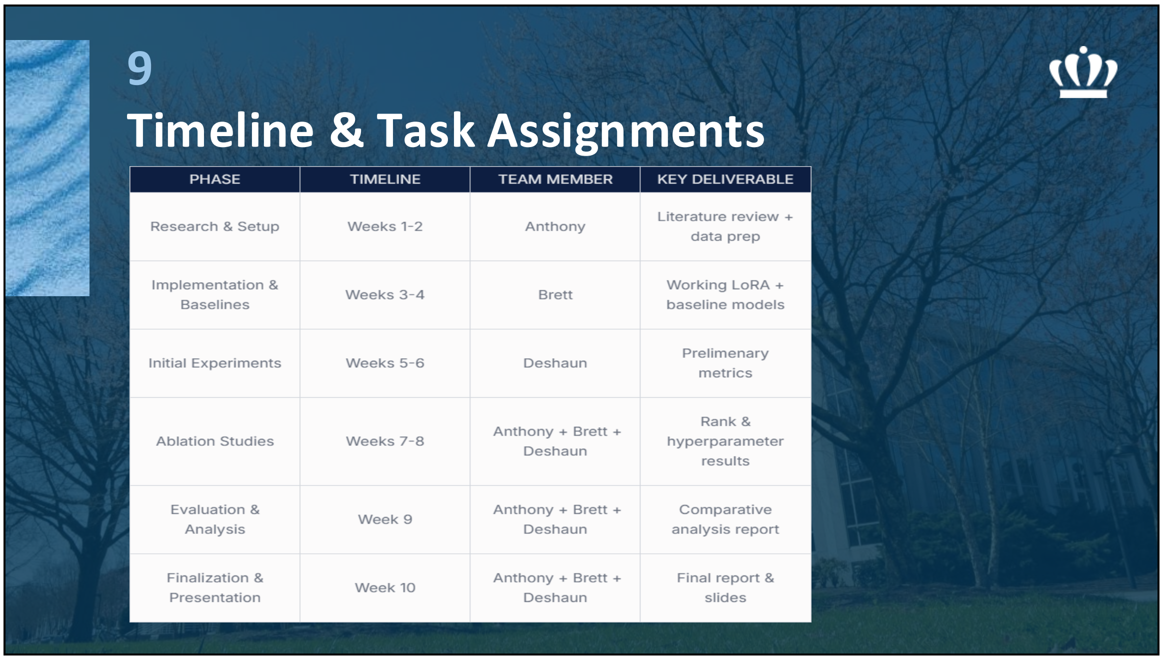
These include full fine-tuning, where every parameter is updated; partial fine-tuning, where only the top few layers are trained; adapters, which add small modules between Transformer layers; and bias-only tuning, or BitFit, which updates only bias terms.

We’ll assess these methods using both performance and efficiency metrics.

For classification tasks from GLUE, we’ll report accuracy or overall GLUE score. For text generation tasks like the E2E NLG Challenge, we’ll use BLEU, ROUGE-L, METEOR, and CIDEr.

On the efficiency side, we’ll track the number of trainable parameters, GPU memory usage, and training throughput, since LoRA’s main promise is high performance at low cost.

Altogether, these comparisons will let us quantify not just how well LoRA performs, but how efficiently it gets there.



9

Timeline & Task Assignments

PHASE	TIMELINE	TEAM MEMBER	KEY DELIVERABLE
Research & Setup	Weeks 1-2	Anthony	Literature review + data prep
Implementation & Baselines	Weeks 3-4	Brett	Working LoRA + baseline models
Initial Experiments	Weeks 5-6	Deshaun	Prelimenary metrics
Ablation Studies	Weeks 7-8	Anthony + Brett + Deshaun	Rank & hyperparameter results
Evaluation & Analysis	Week 9	Anthony + Brett + Deshaun	Comparative analysis report
Finalization & Presentation	Week 10	Anthony + Brett + Deshaun	Final report & slides

Here's our ten-week project timeline and task breakdown.

In the first 2 weeks of October, we'll completed our literature review, familiarized ourselves with the LoRA codebase, and finalized dataset and model selections.

Weeks the last 2 weeks of October we will focus on implementation — setting up LoRA and all our baselines, including full fine-tuning and adapter methods.

During November, we'll run our first experiments on the GLUE classification tasks and the E2E NLG Challenge dataset to verify reproducibility.
Then we'll conduct ablation studies, varying the LoRA rank and layer choices to analyze performance–efficiency trade-offs.

In December will be dedicated to evaluating results and comparing them directly with the original LoRA paper.
Finally, we'll prepare our report and presentation. All members will collaborate to synthesize findings and finalize our conclusions.

Heading Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urnunc vivrra imperdiet enim. Fusce est. Vivamus a tellus.

Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar

Heading Placeholder

Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urnunc

- *Lorem ipsum dolor sit amet, consectetuer*
- *Maecenas porttitor congue massa.*
- *Fusce posuere, magna sed pulvinar*
- *ultricies, purus lectus malesuada libero,*

Heading Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros

Callout 1

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue Maecenas porttitor congue

Callout 2

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue

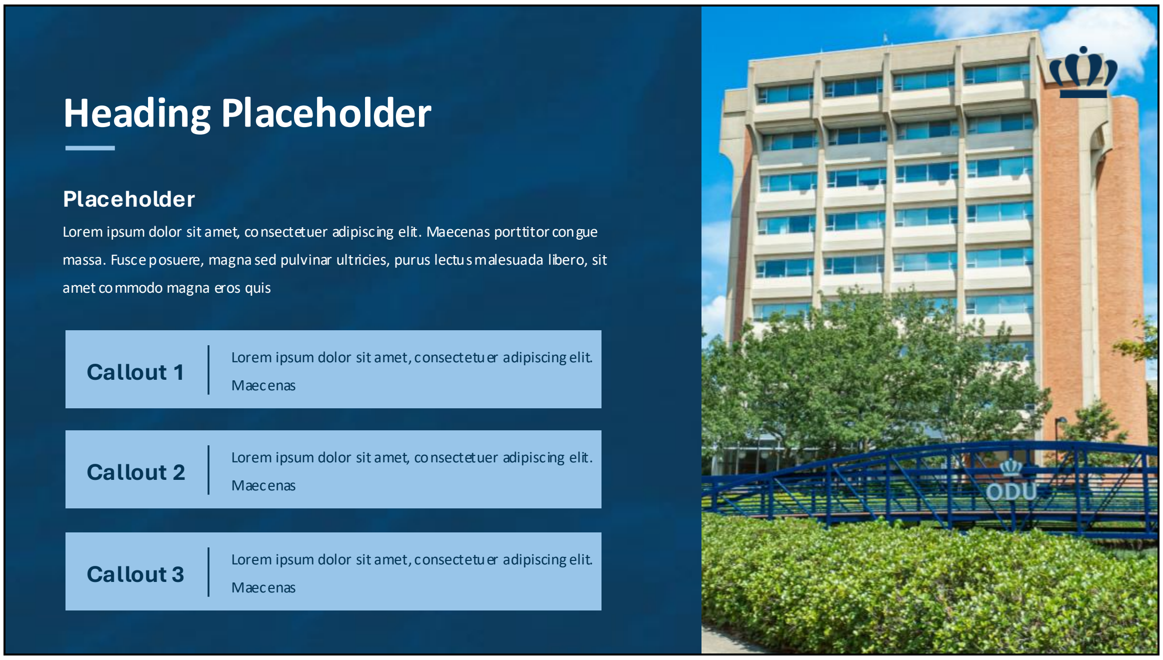
Heading Placeholder

Subhead

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue Maecenas porttitor congue

Subhead

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue Maecenas porttitor congue

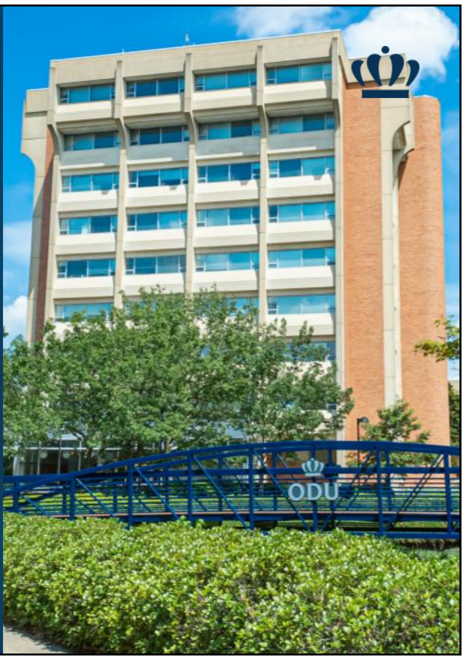


Heading Placeholder

Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis

Callout 1	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas
Callout 2	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas
Callout 3	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas



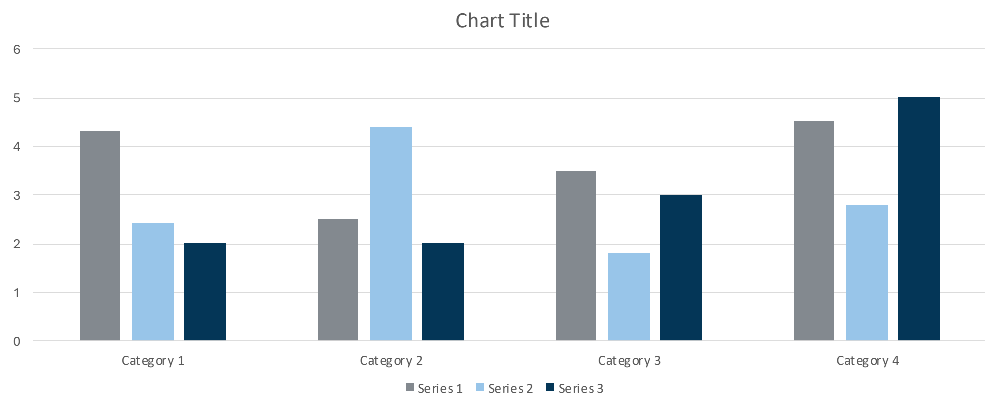
Heading Placeholder

Placeholder

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet

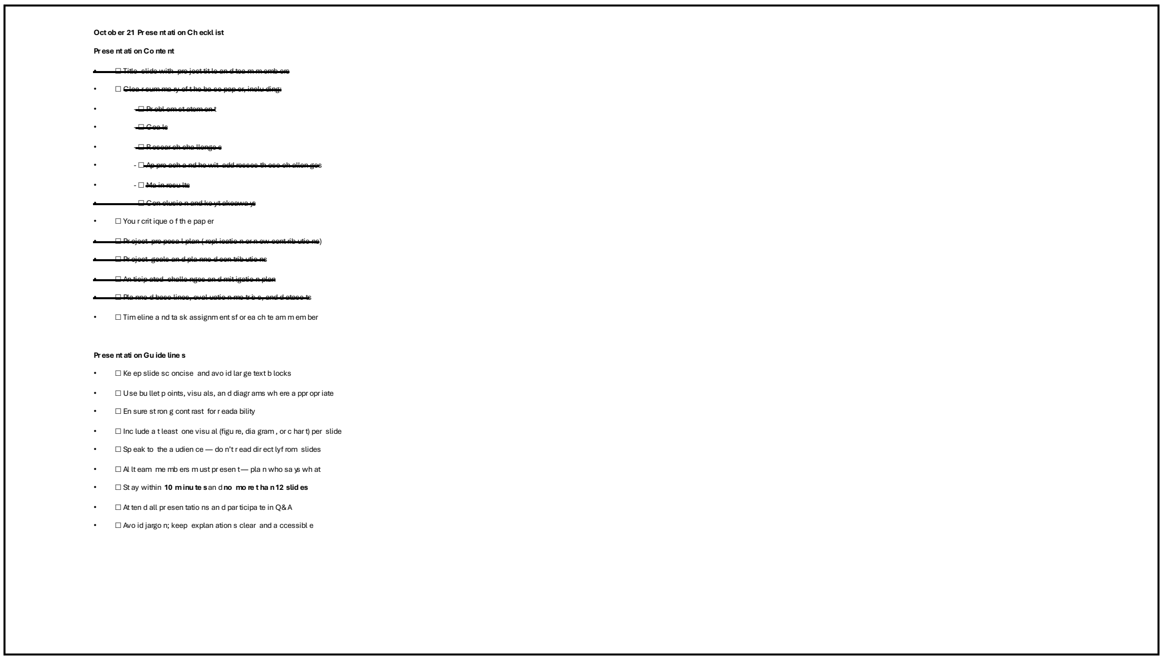
- *Lorem ipsum dolor sit amet, consectetuer*
- *Maecenas porttitor congue massa.*
- *Fusce posuere, magna sed pulvinar*

Chart Header



OLD DOMINION UNIVERSITY

THANK YOU



October 21 Presentation Checklist

Presentation Content

- ~~☐ Title slide with project title and team members~~
- ☐ ~~Clear summary of the base paper, including:~~
- ~~- ☐ Problem statement~~
- ~~- ☐ Goals~~
- ~~- ☐ Research challenges~~
- - ☐ ~~Approach and how it addresses these challenges~~
- - ☐ ~~Main results~~
- ~~☐ Conclusion and key takeaways~~
- ☐ Your critique of the paper
- ~~☐ Project proposal plan (replication or new contributions~~)
- ~~☐ Project goals and planned contributions~~
- ~~☐ Anticipated challenges and mitigation plan~~
- ~~☐ Planned baselines, evaluation metrics, and datasets~~
- ☐ Timeline and task assignments for each team member

Presentation Guidelines

- ☐ Keep slides concise and avoid large text blocks
- ☐ Use bullet points, visuals, and diagrams where appropriate
- ☐ Ensure strong contrast for readability
- ☐ Include at least one visual (figure, diagram, or chart) per slide
- ☐ Speak to the audience — don't read directly from slides
- ☐ All team members must present — plan who says what
- ☐ Stay within **10 minutes** and **no more than 12 slides**
- ☐ Attend all presentations and participate in Q&A
- ☐ Avoid jargon; keep explanations clear and accessible