Final Project: Complete Student Guide

A comprehensive roadmap for choosing, planning, and executing a standout NLP project

---

1. Final Project Overview

The **Final Project** serves as a capstone experience, allowing students to demonstrate and apply the skills they have developed throughout the course. Participants can choose between two pathways:

1. **Default Final Project** – Implement a simplified version of **GPT-2**, then apply the model to three downstream NLP tasks.
2. **Custom Final Project** – Design your own project exploring the intersection of **human language and deep learning**, subject to instructor approval.

---

2. Key Project Details

Grading Breakdown

The final project evaluation consists of multiple components:

- **Project Proposal (5%)** – Outline your objectives, methodology, and expected outcomes.
- **Project Milestone (5%)** – Submit a progress update demonstrating intermediate results.
- **Project Report (35%)** – A comprehensive write-up detailing implementation, results, and analysis.
- **Project Implementation (40%)** – The actual code, model, and technical implementation.
- **Project Presentation (15%)** – Oral presentation of your work to the class.

Deadlines

All submissions (**proposal, milestone, and final report**) are due by **11:59 PM** on their respective dates.

Important Timeline

Deliverable	Due Date	Percentage
Project Proposal	17/4/2025	5%
Project Milestone	4/5/2025	5%
Project Presentation	13/5/2025	15%
Final Report & Implementation	13/5/2025	75%

3. Project Selection: Default vs. Custom

Option 1: Default Project (Language Model Implementation)

- **Objective**: Build a lightweight transformer-based language model (starter code provided) and adapt it for:
  - **Sentiment analysis** (e.g., product reviews).
  - **Creative text generation** (e.g., poetry or paraphrasing).
  - **Efficiency challenges** (e.g., pruning/quantization).
- **Best for**: Students who prefer structure or are new to research.

Option 2: Custom Project

- **Requirements**: Must involve **neural networks + human language** (NLP tasks, model analysis, etc.).
- **Project Archetypes**:
  - **Application-focused**: Solve a real-world task (e.g., fake news detection) using existing models.
  - **Architecture-focused**: Implement/extend a complex model (e.g., sparse transformers).
  - **Analysis-focused**: Investigate model behavior (e.g., bias, linguistic capabilities).
  - **Theoretical/Linguistic**: Prove properties of models/data representations.
- **Idea Generation**:
  - **"Nails" Approach**: Start with a problem (e.g., low-resource translation) and seek solutions.
  - **"Hammers" Approach**: Start with a technique (e.g., contrastive learning) and find new applications.

Resources

- ACL Anthology - Comprehensive repository of NLP research papers.
- arXiv NLP - Preprint server with latest research in computational linguistics.
- Machine Learning Mastery Datasets - Curated list of datasets for various NLP tasks.
- NLP Datasets GitHub Repository - Collection of freely available NLP datasets.
- LLMs from Scratch - Tutorial repository for building language models step-by-step.

---

4. Planning & Execution

Proposal (5% of grade)

- **Elements**:
  - **Literature Review**: Summarize 1–2 key papers; critique their methods/gaps.

  - **Plan**: Define tasks, data sources, baselines, and evaluation metrics (e.g., accuracy, BLEU).
  - **Innovation**: How will you extend prior work? Example: *"Combine prompt engineering with few-shot learning for rare-language tasks."*
- **Length**: 3–4 pages.

Milestone (5%)

- **Expectations**:
  - Working code (even if minimal).
  - Preliminary results (e.g., baseline scores or ablation studies).
- **Tip**: Use synthetic/tiny datasets early to debug (4–8 examples).

Final Report (35%)

- **Structure**:
  1. Abstract
  2. Introduction (motivation, research questions)
  3. Related Work
  4. Methodology (model architecture, data pipeline)
  5. Experiments (hyperparameters, baselines)
  6. Results & Analysis (visualizations, error analysis)
  7. Conclusion & Future Work
- **Pro Tip**: Analyze failures—e.g., *"Our model struggled with negation; future work could integrate syntactic features."*

---

5. Data & Tools

Dataset Sources

General NLP (Benchmark Tasks)

Name	Link	Description
HuggingFace Datasets	Link	50k+ datasets for tasks like text classification, QA, and generation. Use `load_dataset()` for easy access.
GLUE	Link	9 tasks (e.g., sentiment, paraphrase detection) for evaluating model generality.
SQuAD	Link	100k+ QA pairs from Wikipedia. Ideal for custom QA projects.
SNLI	Link	570k sentence pairs labeled for entailment, contradiction, or neutrality.
CoNLL-2003	Link	Named Entity Recognition (NER) dataset for English/German.

Specialized Tasks

Name	Link	Description
SST-2	Link	Stanford Sentiment Treebank - sentences with binary sentiment labels.
CNN/DailyMail	Link	300k news articles for summarization tasks.
MultiWOZ	Link	Task-oriented dialogue dataset (restaurant booking, hotel reservations).
Universal Dependencies	Link	200+ treebanks for dependency parsing in 100+ languages.
WMT	Link	Benchmark datasets for machine translation (e.g., English-German).
LibriSpeech	Link	1k hours of audiobooks for speech-to-text projects.

Custom Data Collection

Name	Link	Description
Twitter/X API	Link	Scrape tweets for real-time sentiment/trend analysis.
Common Crawl	Link	Petabyte-scale web crawl data for training custom corpora.
Amazon Reviews	Link	Multilingual product reviews for sentiment/theme analysis.

Compute Constraints

Avoid Training Large Models From Scratch

- **Examples**: GPT-2 (1.5B), T5 (11B), Llama 2 (7B).
- **Why**: High computational costs and extensive GPU time requirements.

Recommended Strategies

1. **Fine-Tune Pretrained Models**:
   - **Models**: BERT, RoBERTa, T5-Small, Llama 3.
   - **How**: Use HuggingFace's `Trainer` with `peft` for parameter-efficient tuning.
2. **Distillation**:
   - **Tools**: HuggingFace DistilBERT, TensorFlow Model Optimization.
   - **Example**: Distill larger models into smaller variants with `knowledge_distillation_loss`.
3. **Quantization**:
   - **Tools**: ONNX Runtime, PyTorch Quantization.
   - **How**: Convert models to 8-bit/4-bit with `bitsandbytes` for 4x memory reduction.
4. **Leverage APIs**:
   - **Commercial LLMs** (OpenAI, Claude): Use for zero-shot baselines or synthetic data generation.
   - **Cloud NLP Services**: Preprocess text (entity extraction, sentiment) before fine-tuning.

6. Methodology & Debugging

Experimental Design

- **Data Splits**:
  - **Train** (70%) → **Validation** (15%, for hyperparameters) → **Dev** (10%) → **Test** (5%, final eval).
  - **Golden Rule**: Never touch the test set until the end!
- **Overfitting Fixes**:
  - Dropout (start with `p=0.5`).
  - Early stopping (monitor validation-set loss).
  - Regularization techniques (L2, weight decay).

Debugging Tips

1. **Start Simple**: Get a 100% train-set accuracy on 10 examples before scaling.
2. **Incremental Adds**: Add one feature at a time (e.g., attention heads).
3. **Visualize**: Plot attention weights or gradient flows to spot issues.
4. **Systematic Testing**: Create unit tests for each component of your pipeline.

---

7. Ethics & Collaboration

- **Credit**: Document all reused code/data. Grading emphasizes *your* contributions.
- **Teams**: Clearly define roles (e.g., "Alice handled data preprocessing; Bob ran experiments").
- **Model Biases**: Evaluate your model for potential biases and document your findings.

---

8. Tools for Implementation

For Default Final Project (Language Model + NLP Tasks)

1. **HuggingFace Transformers**
   - **Purpose**: Fine-tune language models for sentiment analysis, paraphrasing, or generation.
   - **How to Use**:

```
from transformers import AutoModelForSequenceClassification, Trainer
model = AutoModelForSequenceClassification.from_pretrained("gpt2")
trainer = Trainer(model=model, args=training_args, train_dataset=datase
t)
trainer.train()  # Fine-tune for sentiment analysis
```

   - **Tip**: Use `bitsandbytes` for 4-bit quantization: `load_in_4bit=True` reduces memory usage.
2. **Weights & Biases (W&B)**

- **Purpose**: Track training metrics, attention patterns, and hyperparameters.
- **How to Use**:

```
import wandb
wandb.init(project="nlp-sentiment")
wandb.log({"loss": loss, "accuracy": accuracy})
```

- **Guideline**: Visualize attention heads to debug why paraphrasing fails.

3. **ONNX Runtime**

- **Purpose**: Deploy models efficiently (e.g., as a web API).
- **How to Use**:

```
python -m transformers.onnx --model=gpt2 --feature=sequence-classificat
ion onnx_model/
```

- **Tip**: Quantize further with `onnxruntime.quantize_dynamic()` for edge devices.

4. **PyTorch Lightning**

- **Purpose**: Modularize training loops for downstream tasks.
- **Guideline**: Use `LightningModule` to separate sentiment training from generation logic.

For Custom Final Project (NLP + Deep Learning)

5. **Streamlit**

- **Purpose**: Build UIs for interactive demos (e.g., real-time translation apps).
- **How to Use**:

```
import streamlit as st
st.text_input("Enter text")
st.write(model.generate(text))  # Demo the custom model
```

- **Tip**: Add `st.progress()` to show inference speed for efficiency-focused projects.

6. **Chainlit**

- **Purpose**: Create conversational AI agents (e.g., chatbots).
- **Guideline**:

```
@cl.on_message
async def main(message: str):
    response = custom_model(message)
    await cl.Message(content=response).send()
```

- **Use Case**: Build interactive dialogue systems with low latency.

7. **Evaluation Tools**

- **Purpose**: Compare your model against state-of-the-art baselines.
- **Options**:

- LM Evaluation Harness
- HELM
- ROUGE for summarization
- BERTScore for text generation

8. **HuggingFace Datasets**
   - **Purpose**: Access 50k+ datasets (e.g., SQuAD for QA, GLUE for NLU).
   - **Guideline**:

```
from datasets import load_dataset
dataset = load_dataset("sst2")  # For sentiment analysis
```

Development Workflow

Stage	Tools	Key Activities
Planning	Project Documentation, GitHub Issues	Define scope, create timeline, set evaluation metrics
Development	HuggingFace, PyTorch Lightning	Implement models, build data pipelines, fine-tune
Testing	W&B, Unit Tests	Track metrics, debug model behavior, validate outputs
Deployment	ONNX Runtime, FastAPI/Streamlit	Optimize for inference, create demo interfaces
Analysis	W&B, Evaluation Tools	Compare to baselines, error analysis, document findings

---

Ethics & Best Practices

- **Model Outputs**: Use toxicity detection libraries like `detoxify` to audit generated text for bias.
- **Responsible AI**: Add guardrails if building user-facing applications:

```
from nemoguardrails import RailsConfig
config = RailsConfig.from_path("./guardrails/")  # Block harmful content
```

- **Documentation**: Provide model cards describing limitations and intended use cases.
- **Data Privacy**: Handle user data responsibly and document data processing steps.

---

9. Pro Tips & Common Pitfalls

Pro Tips

- **Write Early**: Draft your report *while* coding—it reveals gaps in logic.
- **Incremental Progress**: Commit code frequently with descriptive messages.

- **Reproducibility**: Use fixed random seeds and document environment requirements.

Common Pitfalls

- **Scope Creep**: Start simple and add complexity only after basics work.
- **Data Quality Issues**: Always inspect your data before training.
- **Overfitting**: Monitor validation metrics closely.
- **GPU Management**: Use gradient accumulation for larger batch sizes.
- **Evaluation Blind Spots**: Test on diverse examples outside your training distribution.

Final Note: The best projects combine rigor with creativity. Whether you're fine-tuning a model or inventing a new metric, focus on *why* your work matters.

11. Submission Checklist

- [ ] Project proposal (3-4 pages)
- [ ] Code repository (well-documented)
- [ ] Milestone report with preliminary results
- [ ] Final report (following required structure)
- [ ] Presentation slides
- [ ] Demo (if applicable)