# A Tiny LLM that Generates NLP Research Problem Statements from Abstracts

Mukhil Muruganantham Prakaash – (mmm9280@psu.edu)

## Problem Definition

Build and train a small-scale open-source LLM to generate NLP research problem statements from abstracts of papers. Despite the explosive growth in scientific publications, identifying the core research **problem** within a paper remains a time-consuming task for researchers, especially when scanning large volumes of literature. Abstracts often contain research problems, but it's not always explicitly labeled or easy to extract programmatically. This project aims to explore whether language models (LLMs), fine-tuned on weakly-labeled abstract-to-problem pairs, can automatically extract research problem statements from scientific abstracts. By leveraging transformer-based architectures like GPT-2 and heuristic data labeling techniques, we seek to evaluate the feasibility and effectiveness of building a lightweight system capable of understanding and reproducing the central research question of a given paper

**Dataset Construction**

- To build a dataset for training a language model to extract research problems from abstract, we utilized the Hugging Face Scientific Papers collection, specifically the "arvix" subset. This dataset includes approx. 1.7 million full text scientific articles segmented into abstract, introduction and body_text field.

Extraction strategy: we focus on generating input-output pairs where

Input: the full abstract of an article

Output: A heuristic approximation of the article's central research problem

A lightweight method was employed to extract the problem statement by selecting the first 2 sentences of each abstract. This approach leverages the observation that authors often introduce the research motivation or problem early in the abstract. Although this heuristic trades off precision for simplicity and speed, it provided a quick way to prototype.

Approximately 200 such pairs were manually selected to facilitate fast experimentation. These were serialized into a JSON format to enable easy loading and manipulation during model training.

Tokenization and Dataset Preparation: The pairs were converted into a hugging face dataset object and tokenized using a GPT-2 tokenizer. Inputs and outputs were concatenated into a single sequence to allow autoregressive training. Padding and truncation were applied with a maximum length of 256 tokens to manage memory constraints. A train-test split (90-10) was created to support evaluation.

**LLM Selection & Training Details.**

Model and Tokenizer : we used the GPT-2 small model (gpt2) with its corresponding tokenizer. The eos_token was used as the padding token to maintain compatibility with the model's architecture

Training Configuration : The model was fine-tuned using the Hugging Face Trainer API. Key training parameters included

- Learning rate: 5e-5
- Batch size: 4 (for both training and evaluation)
- Epochs: 5
- Weight decay: 0.01
- Checkpointing: Disabled (save_strategy="no") to conserve disk space
- Evaluation during training enabled
- Logging every 10 steps

This configuration ensured efficient training on limited hardware while allowing for basic evaluation and debugging.

**Rationale Behind Model Choice**

GPT-2 small was selected due to its:

- Fast training capability on small datasets
- Adequate performance for language modeling tasks
- Compatibility with Hugging Face libraries

Though small in size, it serves as a reliable baseline model for early-stage experimentation.

## Evaluation Metrics & Experiments

Quantitative Evaluation : perplexity

After training, the model was evaluated on the held-out test set using **perplexity**, a standard metric for language modeling. Perplexity is derived from the average negative log-likelihood per token and reflects how "surprised" the model is by the correct output. Lower perplexity indicates better fluency and generalization.

Qualitative Evaluation: Text Generation

To assess generation quality, a pipeline was set up to generate outputs from prompts. For instance:

- Prompt: *Abstract: In this paper, we propose a novel method for improving neural machine translation. Research Problem:*
- Desired behavior: The model generates a coherent and relevant research problem.

We qualitatively evaluated the coherence, specificity, and clarity of these generations. Generated outputs were examined for:

- Relevance to the abstract
- Semantic alignment with typical research problem formulations
- Absence of generic or repetitive phrasing

## Reflections & Lessons Learned

Data Quality Trade-offs: While the first-two-sentence heuristic was quick to implement, it often missed abstracts where the problem is stated later or ambiguously. A more reliable method could involve:

- Discourse parsing
- Manually annotated gold standard datasets
- Extractive summarization techniques

Improved target quality would likely yield better training signals.

Small-Scale Prototyping vs. Real-world Scaling: Prototyping on 200 examples enabled rapid iteration and debugging. However, for a production-level system, a significantly larger and more diverse dataset would be required. Improved data diversity, stratification by domain/year, and higher-quality labels are essential for broader generalizability.

Limitations of Metrics: Perplexity primarily measures fluency, not semantic accuracy. A model might produce fluent text that lacks relevance or substance. Thus, additional metrics such as ROUGE or BLEU, while imperfect, can help assess content overlap. Ultimately, human evaluation remains crucial for assessing whether generated problem statements align with the abstract's core intent.

Model Overfitting: Training on a small dataset with a large-capacity model introduces overfitting risks. Low evaluation loss relative to training loss indicates over-specialization. Regularization techniques and careful monitoring of loss trajectories are needed to manage this.

## Next Steps & Future Work

Several directions emerged from this initial prototype:

Data Improvements

- Curate a high-quality subset where problem statements are clearly defined.
- Explore weak supervision or crowd-sourcing to annotate problem statements.
- Use advanced methods (e.g., extractive summarization, citation context) to derive better supervision signals.

Model & Training Enhancements

- Compare model architectures: decoder-only (GPT) vs. encoder-decoder (T5/BART).
- Experiment with larger models (e.g., GPT-Neo or GPT-J) and observe scaling effects.
- Conduct hyperparameter sweeps to find optimal training regimes.

Evaluation Extensions

- Incorporate ROUGE/BLEU to compare against heuristic targets.
- Include domain experts for human judgment evaluation.
- Measure interpretability and explainability of generated research problems.

User-Centric Applications

- Develop tools to assist researchers in summarizing or formulating their research problems.
- Provide interactive feedback for writing or refining abstracts during submission.