Optimizing Prompts

Prompt Optimization

- Bad prompts can lead to bad performance
- Automatic optimization methods instead of human optimization
- Exists for discrete and continuous prompts

Discrete Prompts

- Sequence of natural language tokens
- Huge searchspace
- Optimize prompts for downstream tasks
  - Gradient-based
  - RL-based
  - Edit-based
  - LLM-based

Gradient-Based

- Maximise output likelihood via gradient update
- Replacing prompt token with candidate token for each position
- Very expensive
  - Transform discrete tokens into continuous embeddings for performance
- Flexibel
- Greedy
- AutoPrompt will be explained later

Reinforcement-Learning-Based

- Consider prompt optimization as RL problem
- RLPrompt and Tempera have use limited search space and modify the original prompt
- PRewrite creates new prompts granting more flexibility

Edit-Based

- Gradient- and RL-based approaches have bad performance on larger models or are not feasible on API-Models
- Genetic prompt search method using a language model
- Edits prompts using the cloze task form:
  - "Text with a [MASK] or missing token here."
- Greedy

LLM-Based

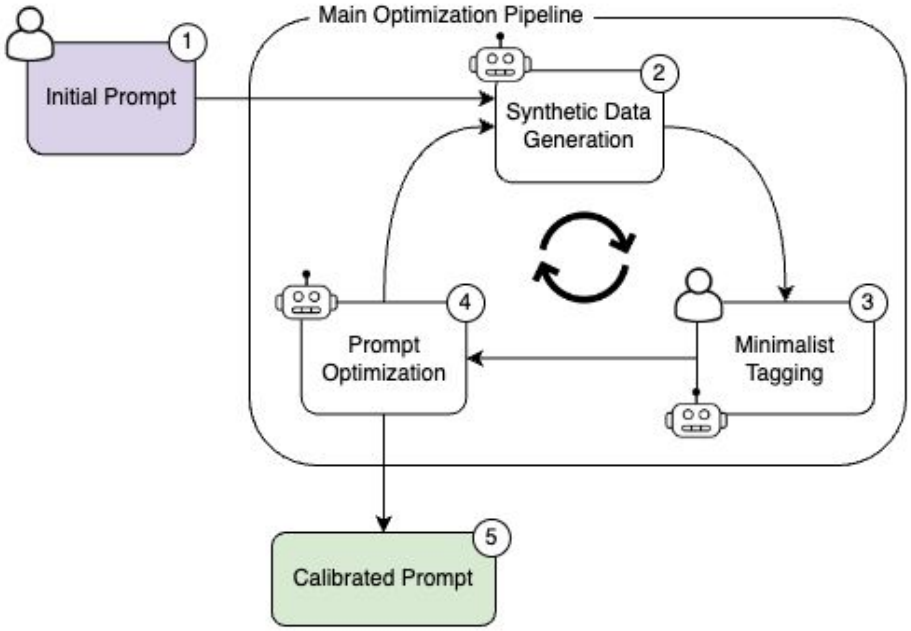
- Use LLMs to create initial prompts
- Improve the best prompt using a iterative Monte Carlo search
- No constraints on the search space may lead to unstable results
- Use previous prompts to progressively generate better results
- Approach is still struggling

Autoprompt

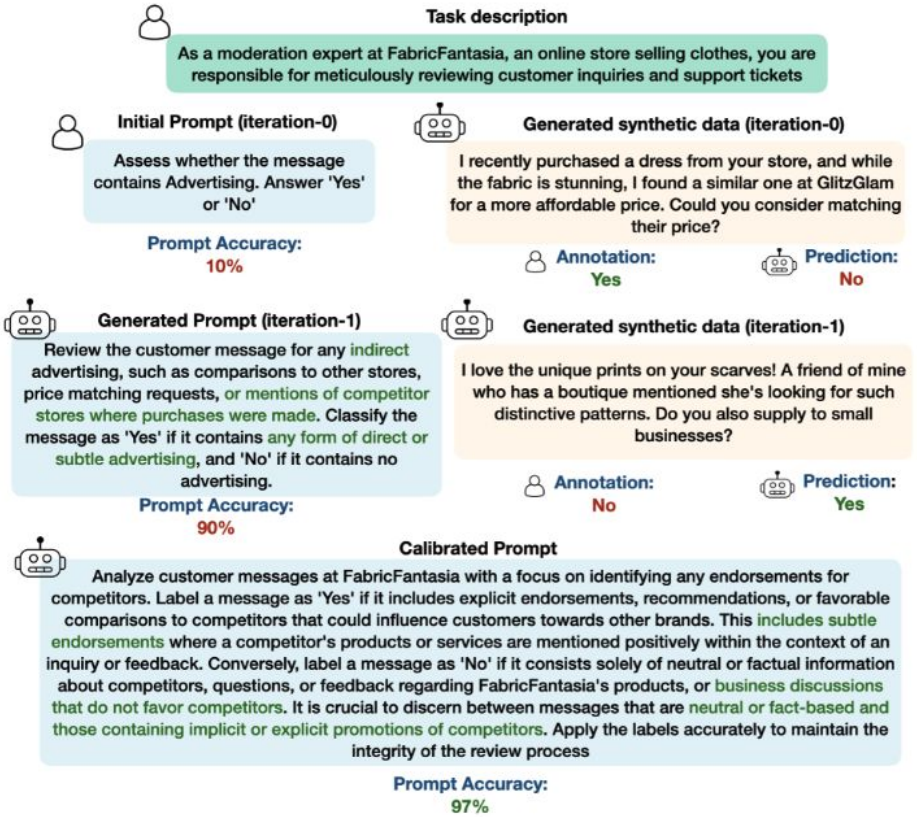
- Goal: **optimize prompt** to optimize **performance** & **accuracy**
  - Better performance boost at **various tasks**
  - Avoid ambiguous prompts (really difficult to understand **true intent**)

Autoprompt: Intent-based Prompt Calibration

- Optimized for **real-world use cases**
  - E.g. moderation (unbalanced data distribution)
- **Modular**
  - Every part can be **used separately**
    - Synth. data generation
    - Prompt distillation
- Default:
  - Optimized for **task classification**
  - Score function: **accuracy**
  - Error analysis: **confusion matrix** and **prompt misclassifications**



Autoprompt (IPC): Example



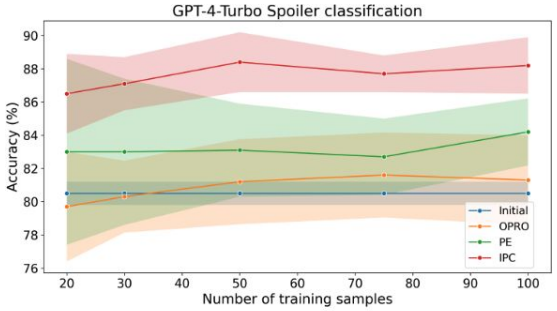
Autoprompt (IPC): Generative Tasks

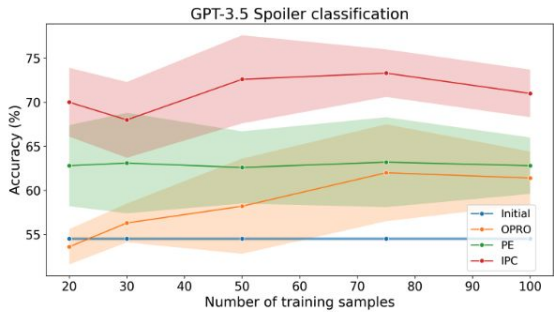
- Split optimization process in **2 parts**
  1. Use LLM to **rephrase initial prompt** to define **ranking task**
     - Treat like **classification task** for classification pipeline
     - Distribution imbalance
       - ➢ **Meta-prompt**: generate challenging boundary samples from the **top two scores**
  2. Use same underlying process to **optimize original generative prompt**
     - Iteratively apply **steps 2 & 3**
     - Score function: calibrated **ranking prompt**
- Classification task => small amount of **annotation effort**

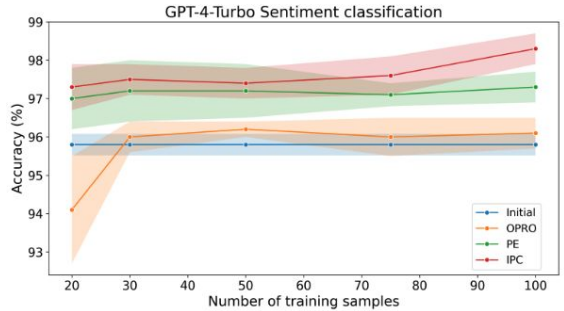
Autoprompt (IPC): Meta-Prompts

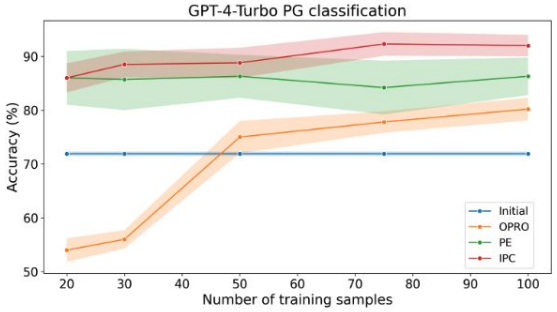
- Consists of **3 separate prompts**
  - **Sample generator**
    - 1st iteration: if no samples provided, instruct model to **generate challenging samples** with even class distribution
    - Next iterations: extended with
      - History with prompts and **good adversarial samples** that confused the prompts
      - Set of realistic examples from dataset
  - **Analyzer**
    - Receives **prompt score**, **confusion matrix** (if classification), and **set of errors** in all classes
    - Produce analysis summary of **prompt performances** and **major failure cases**
  - **Prompt generator**
    - Input: list of **previous prompts + scores** & **performance analysis** of last prompt
    - Output: prompt with higher score

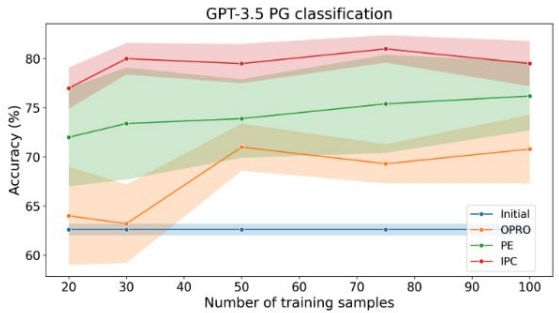
Autoprompt: Accuracy

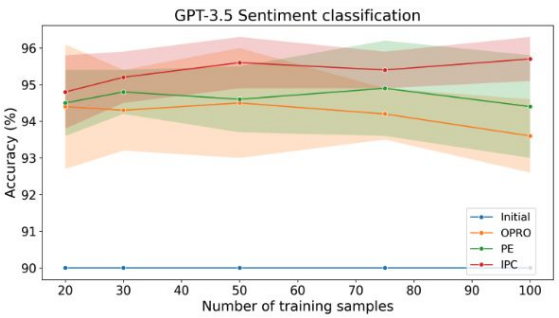


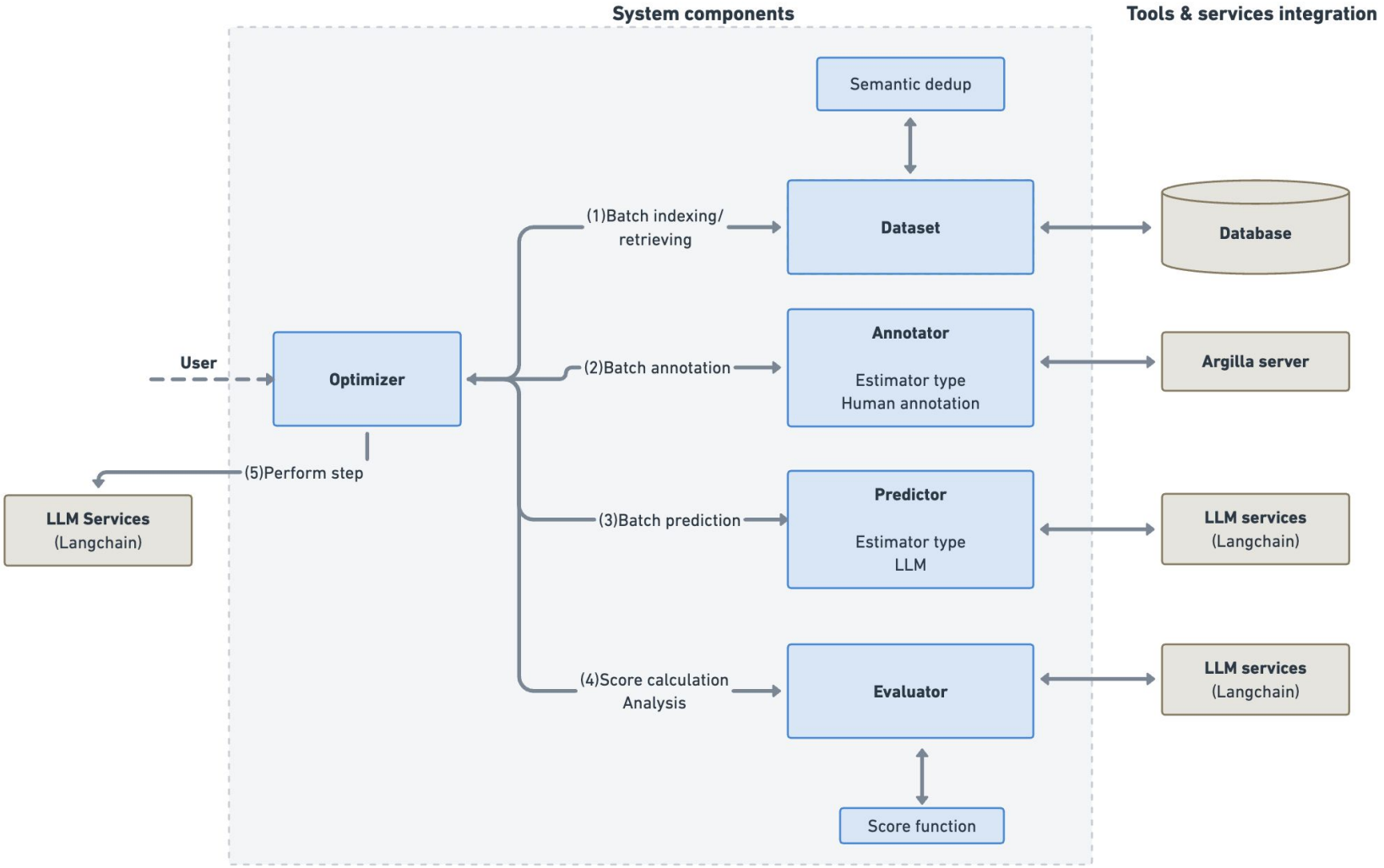











System components
Tools & services integration
Semantic dedup
Dataset
Database
(1)Batch indexing/
retrieving
User
Optimizer
(2)Batch annotation
Annotator
Estimator type
Human annotation
Argilla server
(5)Perform step
LLM Services
(Langchain)
(3)Batch prediction
Predictor
Estimator type
LLM
LLM services
(Langchain)
(4)Score calculation
Analysis
Evaluator
LLM services
(Langchain)
Score function

Autoprompt - Tool

- Easy setup:
  - Git clone
  - Install requirements
  - (install Argilla for human annotation Tool) -> Otherwise LLM
  - Configure API Key and URL
- Configure
  - Annotator:
    - GPT4-o (strong model recommended)
  - Predictor:
    - GPT-4-o-mini (cost efficient model)
  - Meta-Prompt-Model
    - GPT4-o (strong model recommended)

Generation modes

1. Optimization Pipeline
   a. **User Input:** Provide initial prompt and task description.
   b. **Challenging Examples:** Explore difficult cases to improve performance.
   c. **Annotation:** Label examples via human input or LLM.
   d. **Prediction:** Test annotated samples to evaluate performance.
   e. **Prompt Analysis:** Identify and address significant errors.
   f. **Prompt Refinement:** Suggest improved prompts.
   g. **Iteration:** Repeat steps to optimize performance.

2. Generation Pipeline
   a. **User Input**: Provide initial task prompt.
   b. **Prompt Adjustment**: LLM modifies prompt for classification.
   c. **Annotation**: Label challenging examples for classification or ranking.
   d. **Ranker Calibration**: Train LLM-based ranking estimator with annotations.
   e. **Final Calibration**: Use ranker to optimize the original generation prompt.

Filtering Movie Reviews with Spoilers

- Type: Binary-Classification
- Initial: “Does this movie review contain a spoiler? answer Yes or No”
- Task description: “Assistant is an expert classifier that will classify a movie review, and let the user know if it contains a spoiler for the reviewed movie or not.”

- N-examples will be generated of film reviews:
  - Annotated by Human or LLM
- Calibration Process outputs new prompt:

Calibrated prompt

Review Spoiler Identification Protocol: For the task of classifying IMDB reviews for the presence of spoilers, the classifier must label reviews with a heightened sensitivity to nuanced language and indirect spoiler cues. The classification labels are ’Yes’ for spoilers and ’No’ for non-spoilers. Apply the following criteria rigorously: Label ’Yes’ if a review: - Contains subtle references or nuanced language that hints at plot developments or character arcs, without explicit detail. - Includes emotional responses or descriptive language that indirectly reveals plot outcomes or twists. - Employs suggestive language that points to future events or endings, even if it does not reveal specific information. Label ’No’ if a review: - Discusses technical aspects, acting, direction, or personal viewer impressions in a manner that does not hint at or reveal any plot details. - Comments on thematic elements, genre characteristics, or storytelling techniques without disclosing or implying crucial plot twists.

Fact-Classification

- Initial task:I will ask you to assess a scientific claim. Output only the text 'True' if the claim is true, 'False' if the claim is false, or 'NEE' if there's not enough evidence.
  - Accuracy: 35-38 %
- Refined Task after 10 iterations (<10 ct)
  - Implement an advanced classification system for scientific claims using the labels 'True', 'False', and 'NEE'. Assign 'True' to claims that are supported by a robust and consistent scientific consensus, ensuring alignment with current scientific understanding. Label 'False' for claims that have been conclusively debunked with strong evidence, particularly those with a history of scientific refutation, such as perpetual motion machines or debunked medical practices. Use 'NEE' for claims lacking sufficient evidence or those that are speculative but not yet debunked, carefully distinguishing between claims that are truly without evidence and those with evolving scientific discourse. Pay particular attention to speculative or evolving scientific theories, such as the nature of dark energy or quantum entanglement applications, to prevent their misclassification as 'NEE' when they should be 'False'. Enhance accuracy by improving the differentiation between speculative claims and well-supported scientific consensus, and address historical misconceptions to reduce errors in classification. This approach aims to minimize false negatives and false positives in identifying 'True', 'False', and 'NEE' claims, ensuring reliable classification consistent with current scientific standards. Only output the labels 'True', 'False', or 'NEE'.
  - Accuracy: 50-55%

Usage

- Supports different goals: Classification, Generation
- With replaced human-in-the-loop automated
- Quality corresponds with models chosen and “time/money” spent
- “Relative” cost efficient (<1$ per refinement)