TextGrad

TextGrad: Automatic "Differentiation" via Text

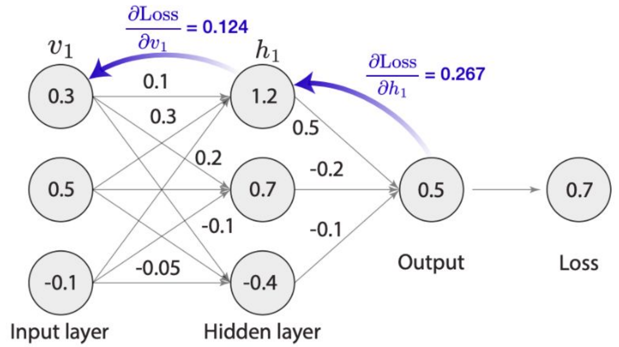
Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, James Zou

San Diego Machine Learning

Abstract

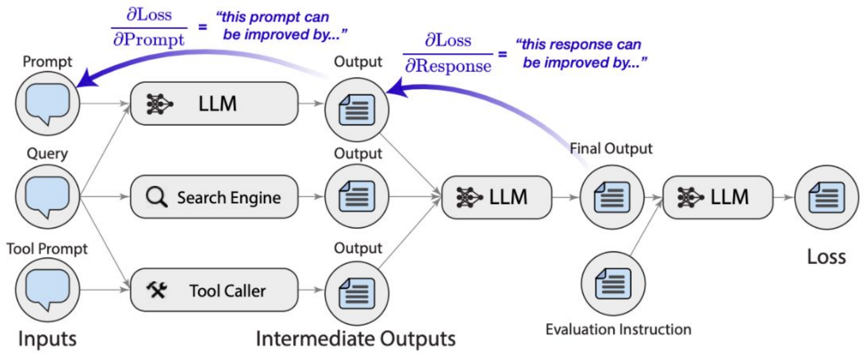
- Large Language Models are extremely powerful but it’s difficult to know if you are using them optimally
- Very simple changes in prompts or input structure can have a big impact on the quality of the response returned by the model
- “Prompt Engineering” has become a common practice to try to get the best result out of models
- Often people iterate and try different prompts until they find something that solves their problem
- An LLM can be plugged in as an optimizer inplace of a human prompt engineer
- TextGrad is a generic framework to allow an LLM to function as an optimizer of graphs similar to how pytorch is a framework for optimizing neural network graphs

a **Neural network and backpropagation using numerical gradients**



$\frac{\partial \text{Loss}}{\partial v_1} = 0.124$
$\frac{\partial \text{Loss}}{\partial h_1} = 0.267$
v_1
h_1
0.3
0.5
-0.1
1.2
0.7
-0.4
0.5
0.7
0.1
0.3
0.2
-0.1
-0.05
0.5
-0.2
-0.1
Output
Loss
Input layer
Hidden layer

b **Blackbox AI systems and backpropagation using natural language 'gradients'**



$\frac{\partial \text{Loss}}{\partial \text{Prompt}}$ = "this prompt can be improved by..."
$\frac{\partial \text{Loss}}{\partial \text{Response}}$ = "this response can be improved by..."
Prompt
Query
Tool Prompt
Inputs
LLM
Search Engine
Tool Caller
Output
Output
Output
Intermediate Outputs
LLM
Final Output
Evaluation Instruction
LLM
Loss

c **❶ Analogy in abstractions**

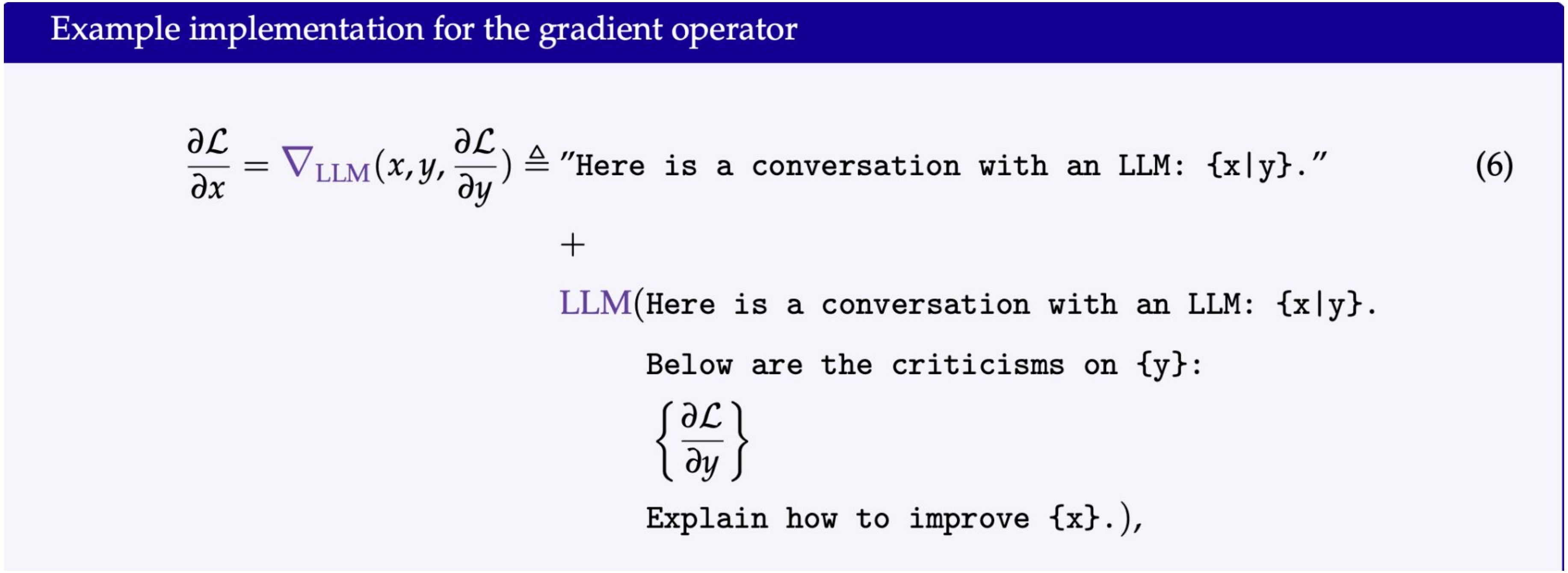
	Math	PyTorch	TextGrad
Input	x	`Tensor(image)`	`tg.Variable(article)`
Model	$\hat{y} = f_\theta(x)$	`ResNet50()`	`tg.BlackboxLLM("You are a summarizer.")`
Loss	$L(y, \hat{y}) = \sum_i y_i \log(\hat{y}_i)$	`CrossEntropyLoss()`	`tg.TextLoss("Rate the summary.")`
Optimizer	$\text{GD}(\theta, \frac{\partial L}{\partial \theta}) = \theta - \frac{\partial L}{\partial \theta}$	`SGD(list(model.parameters()))`	`tg.TGD(list(model.parameters()))`

❷ Automatic differentiation

PyTorch and TextGrad share the same syntax for backpropagation and optimization.

Forward pass	Backward pass	Updating variable
`loss = loss_fn(model(input))`	`loss.backward()`	`optimizer.step()`

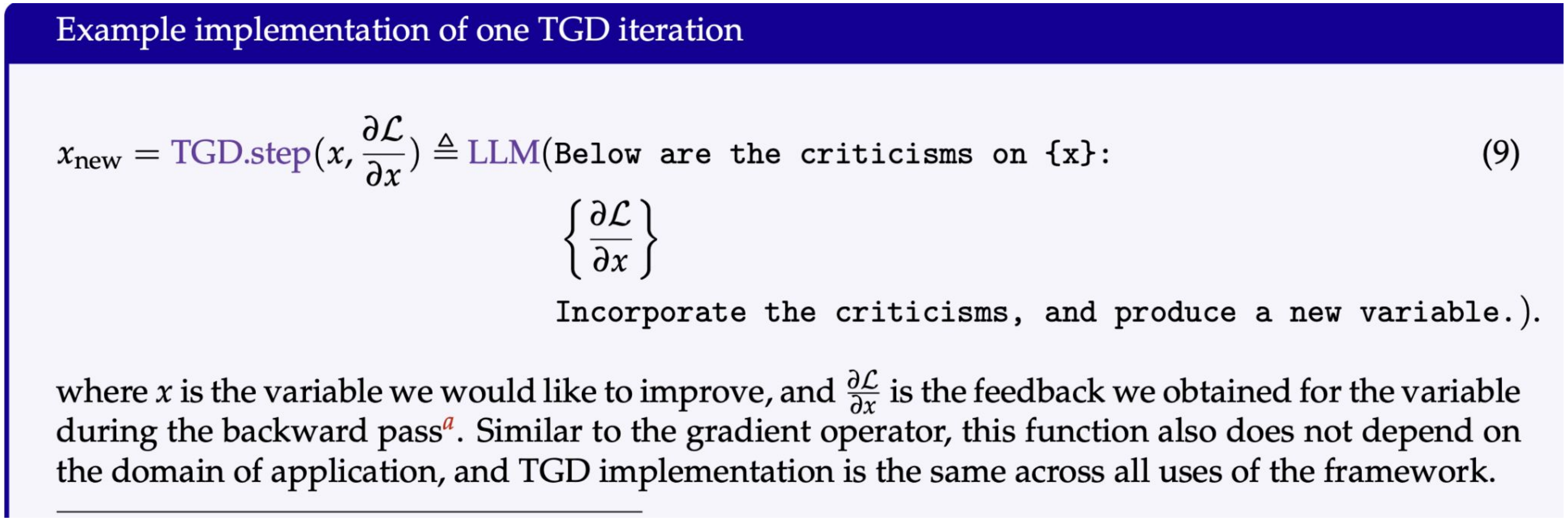
What is a text gradient?



Example implementation for the gradient operator

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x} = \nabla_{\text{LLM}}\left(x, y, \frac{\partial \mathcal{L}}{\partial y}\right) \triangleq\ & \texttt{"Here is a conversation with an LLM: \{x|y\}."} \\
& + \\
& \text{LLM}(\texttt{Here is a conversation with an LLM: \{x|y\}.} \\
& \qquad \texttt{Below are the criticisms on \{y\}:} \\
& \qquad \left\{\frac{\partial \mathcal{L}}{\partial y}\right\} \\
& \qquad \texttt{Explain how to improve \{x\}.}),
\end{aligned} \tag{6}
$$

How to step with text gradients



Example implementation of one TGD iteration

$$x_{\text{new}} = \text{TGD.step}\left(x, \frac{\partial \mathcal{L}}{\partial x}\right) \triangleq \text{LLM}(\texttt{Below are the criticisms on \{x\}:} \left\{\frac{\partial \mathcal{L}}{\partial x}\right\} \texttt{Incorporate the criticisms, and produce a new variable.}). \tag{9}$$

where x is the variable we would like to improve, and $\frac{\partial \mathcal{L}}{\partial x}$ is the feedback we obtained for the variable during the backward pass[a]. Similar to the gradient operator, this function also does not depend on the domain of application, and TGD implementation is the same across all uses of the framework.

LLM("You are an intelligent assistant used as an evaluator, and part of an optimization system. You will analyze a code implementation for a coding problem and unit test results. The code will be tested with harder tests, so do not just check if the code passes the provided tests. Think about the correctness of the code and its performance in harder test cases. Give very concise feedback. Investigate the code problem and the provided implementation. For each failed unit test case, start analyzing it by saying "The code did not pass this test because...". Explain why the current implementation is not getting the expected output. Do not provide a revised implementation. Carefully suggest why there are issues with the code and provide feedback.
{Test-time Instruction}
The coding problem:
{Problem}
Code generated that must be evaluated for correctness and runtime performance
{**Code**}
The test results:
{Local test Results}

Table 1: Code optimization for LeetCode Hard using `gpt-4o`. Results are averaged over 5 seeds.

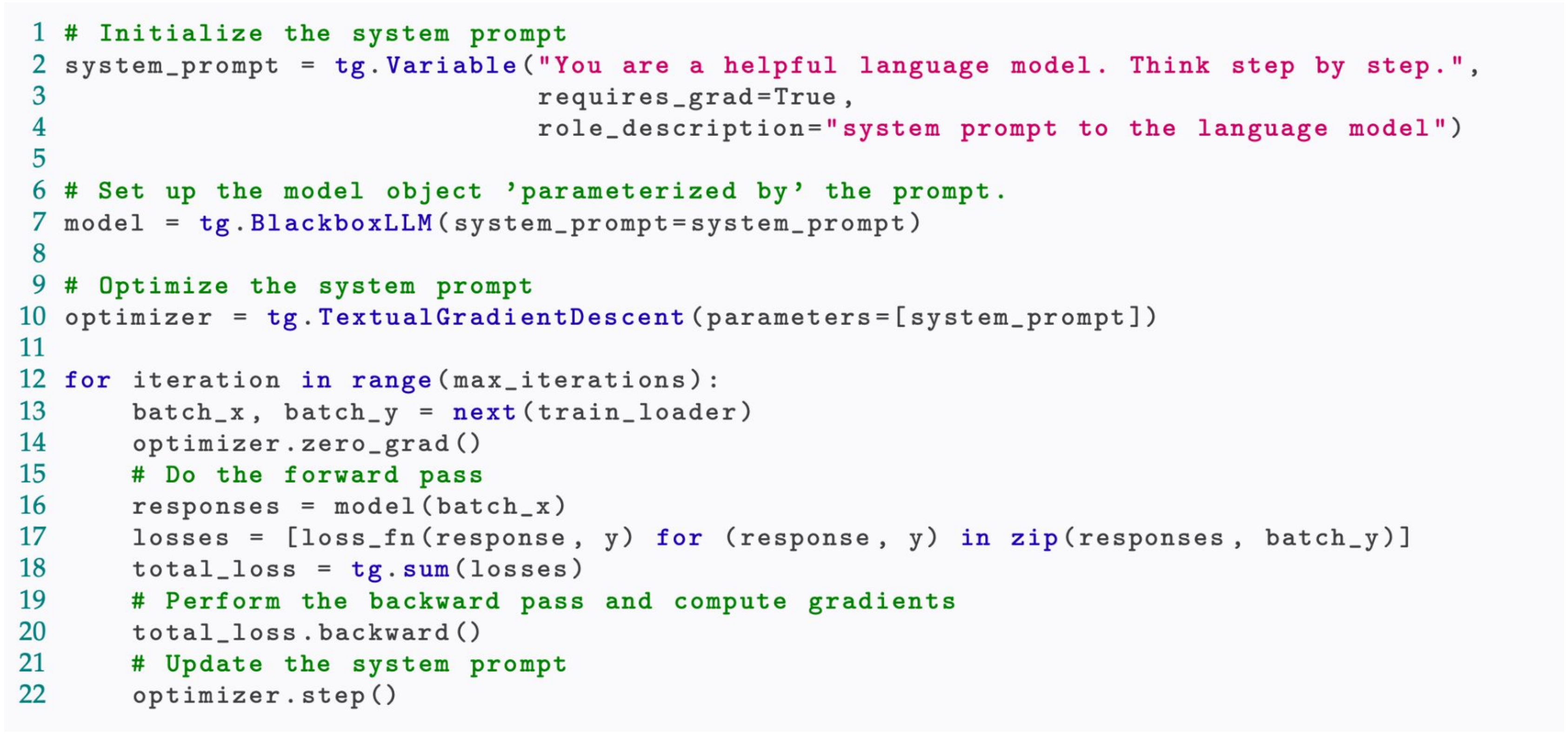
Task	Method	Completion Rate
	Zero-shot [26]	0.26
LeetCode Hard [26]	Reflexion (1 demonstration, 5 iterations) [26]	0.31 ± 0.012
	TEXTGRAD (0 demonstrations, 5 iterations)	$\mathbf{0.36} \pm 0.018$

Instance Optimization

```python
# Assume we have the test_dataset
question, answer = test_dataset[i]
# Initialize the test time loss function
# This has the system prompt provided above as 'Solution Refinement Objective'
test_time_loss_fn = MultipleChoiceTestTime()
# Get a zero-shot solution from an LLM
solution: Variable = zero_shot_llm(question)
# Optimize the solution itself
optimizer = tg.TextualGradientDescent(parameters=[solution])

for iteration in range(max_iterations):
    optimizer.zero_grad()
    # Note how the loss is self-supervised (does not depend on any ground truth.)
    loss = test_time_loss_fn(question, solution)
    # Populate the gradients, and update the solution
    loss.backward()
    optimizer.step()
```

Prompt Optimization



```python
# Initialize the system prompt
system_prompt = tg.Variable("You are a helpful language model. Think step by step.",
                            requires_grad=True,
                            role_description="system prompt to the language model")

# Set up the model object ’parameterized by’ the prompt.
model = tg.BlackboxLLM(system_prompt=system_prompt)

# Optimize the system prompt
optimizer = tg.TextualGradientDescent(parameters=[system_prompt])

for iteration in range(max_iterations):
    batch_x, batch_y = next(train_loader)
    optimizer.zero_grad()
    # Do the forward pass
    responses = model(batch_x)
    losses = [loss_fn(response, y) for (response, y) in zip(responses, batch_y)]
    total_loss = tg.sum(losses)
    # Perform the backward pass and compute gradients
    total_loss.backward()
    # Update the system prompt
    optimizer.step()
```

Results

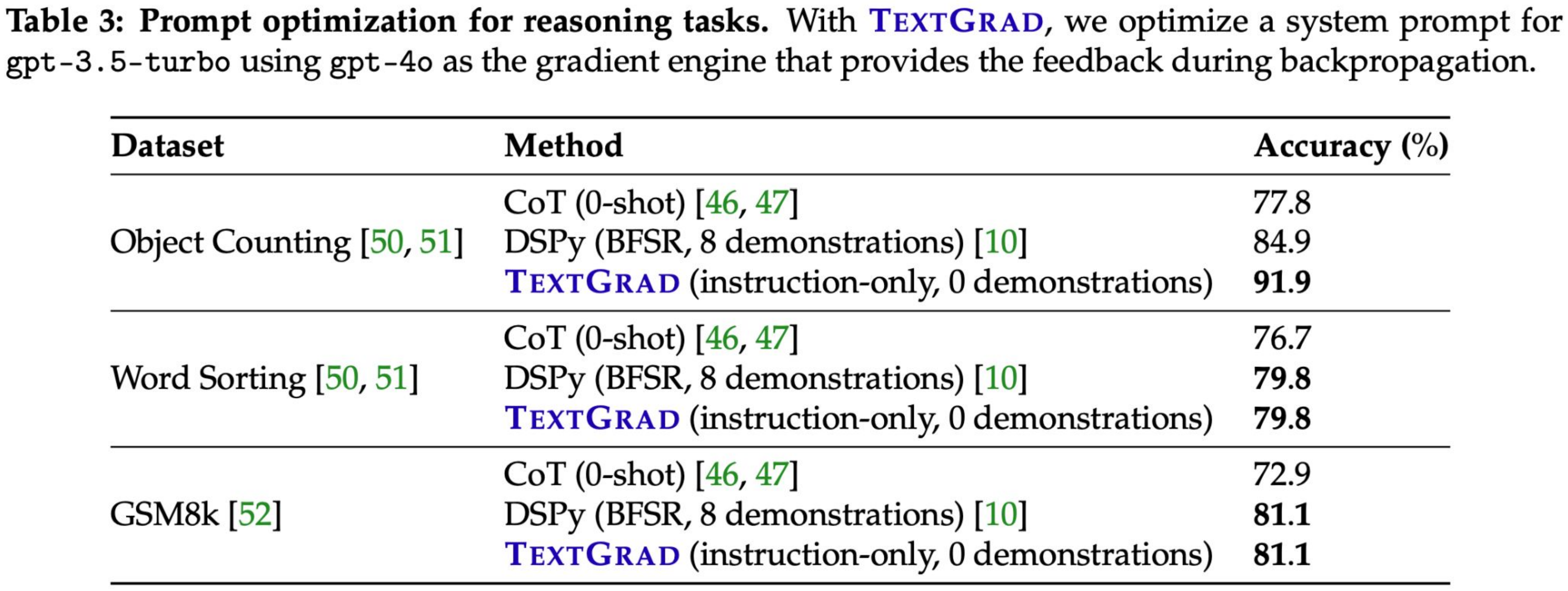
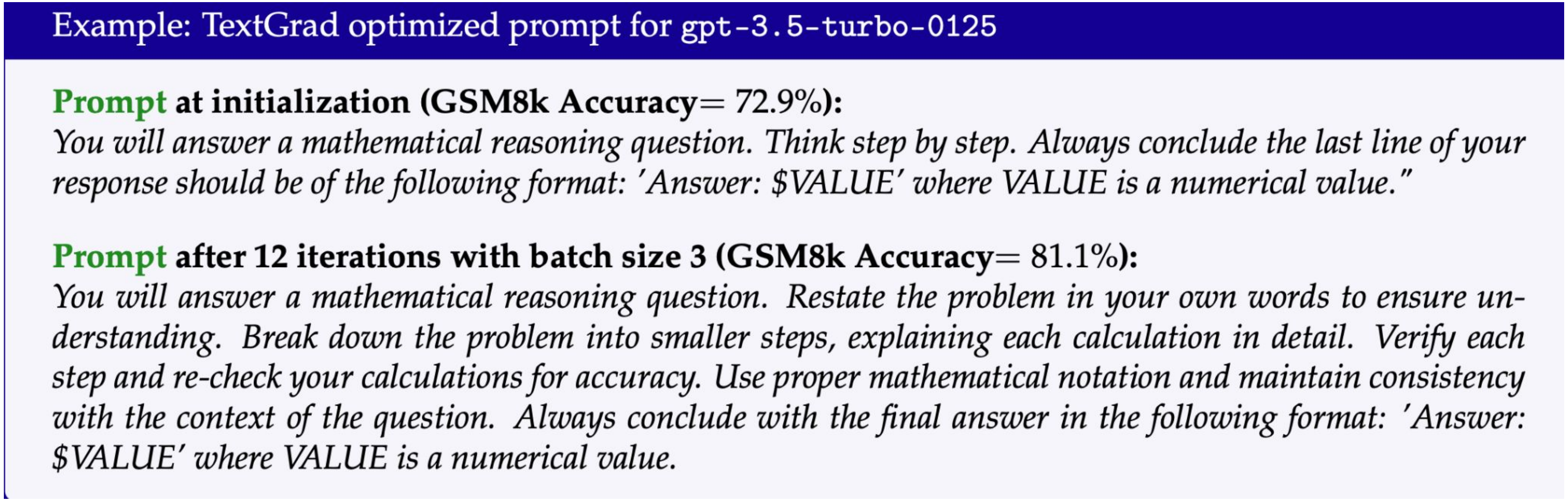


Table 3: Prompt optimization for reasoning tasks. With TEXTGRAD, we optimize a system prompt for `gpt-3.5-turbo` using `gpt-4o` as the gradient engine that provides the feedback during backpropagation.

Dataset	**Method**	**Accuracy (%)**
Object Counting [50, 51]	CoT (0-shot) [46, 47]	77.8
	DSPy (BFSR, 8 demonstrations) [10]	84.9
	TEXTGRAD (instruction-only, 0 demonstrations)	**91.9**
Word Sorting [50, 51]	CoT (0-shot) [46, 47]	76.7
	DSPy (BFSR, 8 demonstrations) [10]	**79.8**
	TEXTGRAD (instruction-only, 0 demonstrations)	**79.8**
GSM8k [52]	CoT (0-shot) [46, 47]	72.9
	DSPy (BFSR, 8 demonstrations) [10]	**81.1**
	TEXTGRAD (instruction-only, 0 demonstrations)	**81.1**

Example Prompt Result

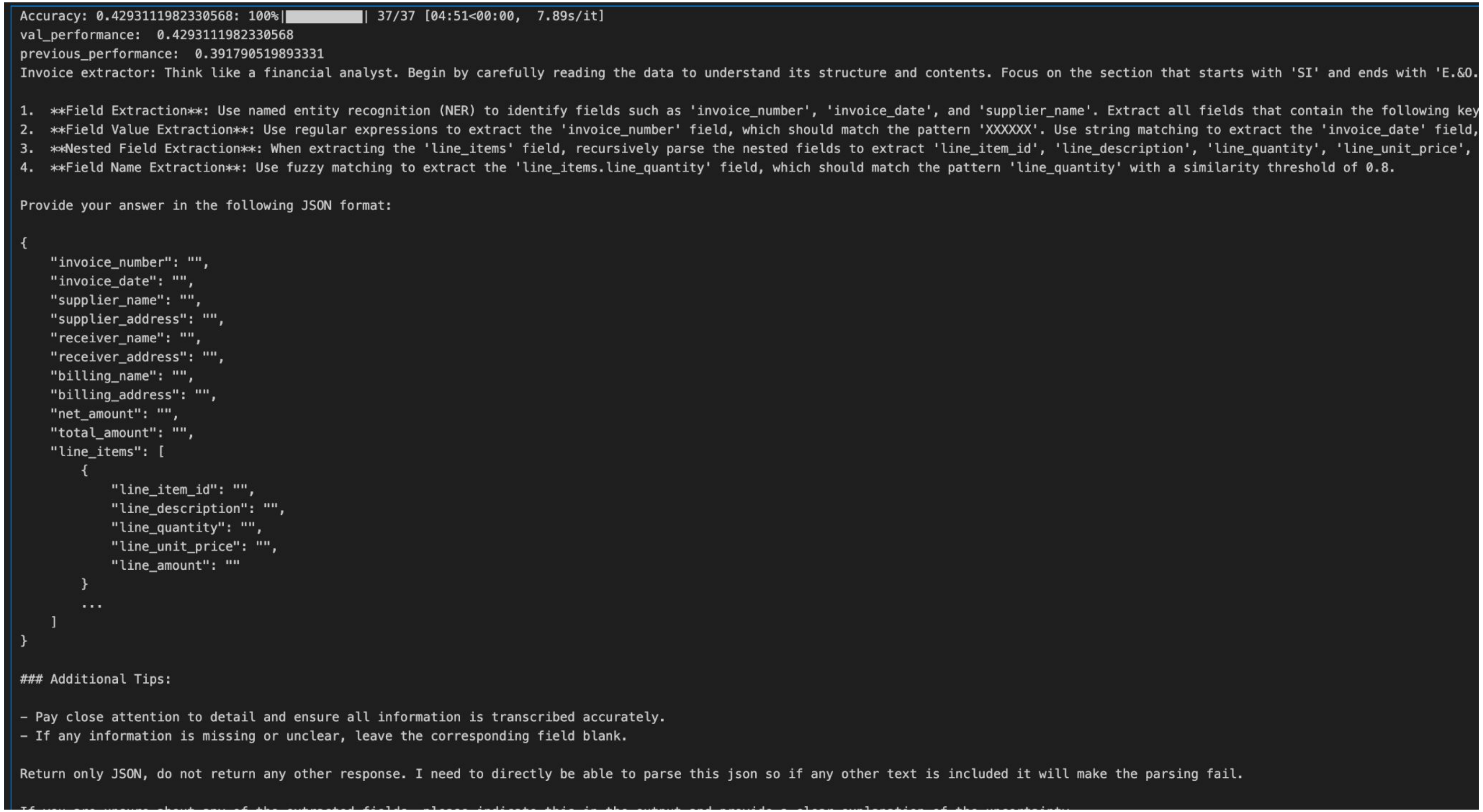


Example: TextGrad optimized prompt for `gpt-3.5-turbo-0125`

Prompt at initialization (GSM8k Accuracy= 72.9%**):**
You will answer a mathematical reasoning question. Think step by step. Always conclude the last line of your response should be of the following format: 'Answer: $VALUE' where VALUE is a numerical value."

Prompt after 12 iterations with batch size 3 (GSM8k Accuracy= 81.1%**):**
You will answer a mathematical reasoning question. Restate the problem in your own words to ensure understanding. Break down the problem into smaller steps, explaining each calculation in detail. Verify each step and re-check your calculations for accuracy. Use proper mathematical notation and maintain consistency with the context of the question. Always conclude with the final answer in the following format: 'Answer: $VALUE' where VALUE is a numerical value.

Live Demo + Questions



```
Accuracy: 0.4293111982330568: 100%|██████████| 37/37 [04:51<00:00,  7.89s/it]
val_performance:  0.4293111982330568
previous_performance:  0.391790519893331
Invoice extractor: Think like a financial analyst. Begin by carefully reading the data to understand its structure and contents. Focus on the section that starts with 'SI' and ends with 'E.&O.

1.  **Field Extraction**: Use named entity recognition (NER) to identify fields such as 'invoice_number', 'invoice_date', and 'supplier_name'. Extract all fields that contain the following key
2.  **Field Value Extraction**: Use regular expressions to extract the 'invoice_number' field, which should match the pattern 'XXXXXX'. Use string matching to extract the 'invoice_date' field,
3.  **Nested Field Extraction**: When extracting the 'line_items' field, recursively parse the nested fields to extract 'line_item_id', 'line_description', 'line_quantity', 'line_unit_price',
4.  **Field Name Extraction**: Use fuzzy matching to extract the 'line_items.line_quantity' field, which should match the pattern 'line_quantity' with a similarity threshold of 0.8.

Provide your answer in the following JSON format:

{
    "invoice_number": "",
    "invoice_date": "",
    "supplier_name": "",
    "supplier_address": "",
    "receiver_name": "",
    "receiver_address": "",
    "billing_name": "",
    "billing_address": "",
    "net_amount": "",
    "total_amount": "",
    "line_items": [
        {
            "line_item_id": "",
            "line_description": "",
            "line_quantity": "",
            "line_unit_price": "",
            "line_amount": ""
        }
        ...
    ]
}

### Additional Tips:

- Pay close attention to detail and ensure all information is transcribed accurately.
- If any information is missing or unclear, leave the corresponding field blank.

Return only JSON, do not return any other response. I need to directly be able to parse this json so if any other text is included it will make the parsing fail.

If you are unsure about any of the extracted fields, please indicate this in the output and provide a clear explanation of the uncertainty.
```