Scaling Textual Gradients via Sampling-Based Momentum

Zixin Ding
University of Chicago
zixin@uchicago.edu

Junyuan Hong
The University of Texas at Austin
jyhong@utexas.edu

Jiachen T. Wang
Princeton University
tianhaowang@princeton.edu

Zinan Lin
Microsoft Research
zinanlin@microsoft.com

Zhangyang Wang
The University of Texas at Austin
atlaswang@utexas.edu

Yuxin Chen
University of Chicago
chenyuxin@uchicago.edu

Abstract

As prompts play an increasingly critical role in Large Language Models (LLMs), optimizing textual prompts has become a crucial challenge. The Textual Gradient Descent (TGD) framework has emerged as a promising data-driven approach that iteratively refines textual prompts using LLM-suggested updates (or textual gradients) over minibatches of training samples. In this paper, we empirically demonstrate that scaling the number of training examples would first enhance and decrease TGD's performance across multiple downstream NLP tasks. However, while data scaling improves results, it also significantly increases computational cost when leveraging LLMs. To address this, we take inspiration from numerical gradient descent and propose Textual Stochastic Gradient Descent with Momentum (TSGD-M)—a method that facilitates scalable in-context learning by reweighting prompt sampling based on past batch distributions. Across 9 NLP tasks spanning three domains—including BIG-Bench Hard (BBH), natural language understanding tasks, and reasoning tasks—TSGD-M significantly outperforms TGD baselines that do not incorporate reweighted sampling, while also reducing variance in most tasks.

1 Introduction

LLMs are highly sensitive to prompt design, with slight variations yielding markedly different outcomes [44, 48, 31, 3]. Automatic Prompt engineering seeks to harness this sensitivity by crafting inputs that maximize LLM capabilities using LLMs' feedback. Recent works [13, 33, 41] propose **Textual Gradient Descent** (TGD) framework for automatically optimizing prompts using LLM itself - a method that closely mirrors numerical gradient descent (GD) in optimization. Just as GD iteratively refines parameters using gradients to minimize a loss function, TGD iteratively refines prompts using "textual gradients"—updates derived from LLM-generated feedback. While many-shot in-context learning can rival fine-tuning by simply increasing demonstrations [1], inference costs grow quadratically with sequence length [11, 35], and LLMs struggle with very long contexts [18, 25]. Given context length constraints, we categorize all of the prior works [13, 33, 47] as **Textual Stochastic Gradient Descent (TSGD)** instead of TGD, which samples *minibatches* of demonstrations stochastically to generate textual gradients—descriptions of current prompts' flaws relative to the sampled minibatch per iteration. This stochastic approach resembles stochastic gradient descent (SGD), allowing for computational feasibility within the limited context length of LLMs.

Stepping back, this raises one question: Do LLMs truly optimize like traditional deep neural networks (DNNs) training? Unlike DNNs, where averaging over the entire dataset removes the stochastic noise

Preprint.

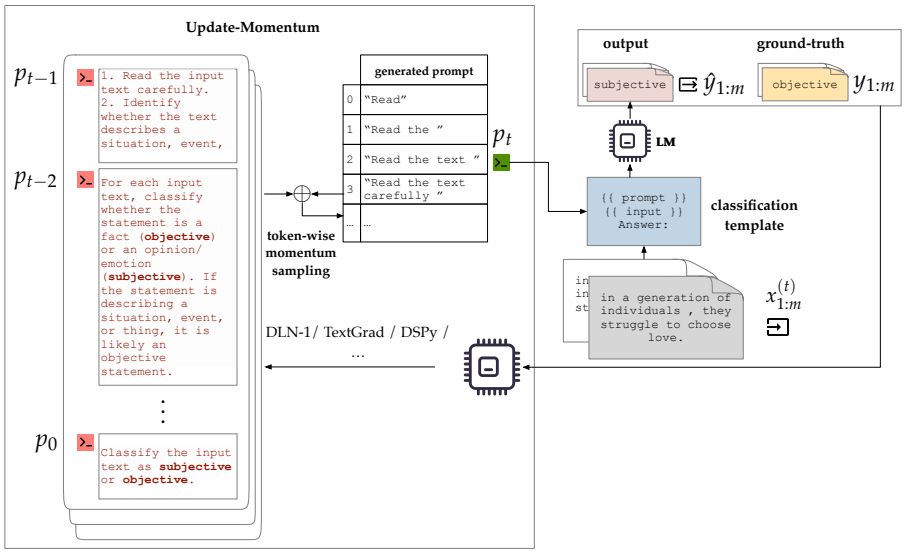


Figure 1: A running example of TSGD-M on the classification task with the Subj dataset [24]. Unlike standard approaches generating prompts solely from the previous meta prompt p_{t-1}, TSGD-M performs momentum-based sampling over all past meta prompts $\{p_\tau\}_{\tau=0}^{t-1}$ per token generation step.

that mini-batches introduce, LLMs may misestimate errors when summarizing flaws over the entire dataset [3]. Previous works highlight LLMs degrading performance in long-context tasks, ranging from multi-document question answering [18] to specification-heavy reasoning [25].

Facing such controversy, we ask: *Is scaling the data all we need in TGD? Do TSGD and SGD share the same inherent property? If so, can we benefit from more advanced optimization techniques from traditional GD optimization literature?*

In this paper, we revisit the impact of data scaling on Textual Gradient Descent (TGD) and Textual Stochastic Gradient Descent (TSGD). Through extensive experiments across 9 NLP tasks, we observe that: (1) moderate scaling of in-context examples per iteration and the total training data used across the pipeline can significantly improve the performance of TGD frameworks such as DSPy [13], DLN1 [33], and TextGrad [47], given sufficient optimization iterations; and (2) beyond a certain point, further increasing the number of in-context examples per iteration or expanding the full training set shall plateau in effectiveness, and in most cases, even lead to performance degradation.

TSGD introduces large variance because of mini-batches and the uncertainty [17] from LLMs. We draw inspiration from GD with momentum [30, 26, 19] to develop TSGD-M (Figure 1). TSGD-M leverage adaptive sampling to reweight past minibatches to smooth textual gradients estimates. We sample from previous textual gradients tokenwise to generate a new pool of prompts for each iteration. An example of TSGD-M optimized prompt under DLN1 framework [33] is shown in Table 1. More optimized prompts samples are presented in Appendix F.4.

Our main contributions are

- We revisit data scaling in TGD and TSGD and draw crucial insights on inputs length for iterative APE tasks performance.
- We devised the first TSGD with momentum method (TSGD-M) which enables extensive data scaling for TSGD with lower variance.
- We provided a theoretical justification to show that TSGD-M achieves lower variance and better training performance (Appendix D), and performed extensive experiments involving different LLMs in a wide range of *Big-Bench Hard* (BBH), NLU and reasoning tasks, demonstrating the performance of TSGD-M against TGD and TSGD.

Approach	Optimized Prompt	Acc.
Human	Classify the input text as subjective or objective.	0.491
DLN1	1. Carefully read the input text. 2. Identify the type of language used in the text. 3. Determine if the text includes words that express the author's opinion, emotion, or perspective. Look for words such as "I", "me", "my", "we", "us", "our", "believe", "think", "feel", "opinion", "perspective", "view", "attitude", "emotion", etc. If it does...	0.713
DLN1-Momentum = 0.6	Classify each input text as subjective or objective. Subjective texts express a personal opinion, emotion, or experience.They often use words and phrases like: - "I think", "I believe", "I feel", "my opinion", "my experience", "I love", "I hate", etc. - Use of first-person pronouns (I, me, my) - Words that describe emotions (e.g., beautiful, sad, excited) - Use of evaluative language (e.g "smart and alert , thirteen conversations about one thing is a small gem .	0.770

Table 1: Prompt template for the **Subj** task, showing Human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance. DLN1-Momentum improves the prompt by keeping the task description after iterative refinement.

2 Preliminaries and Related Work

Problem Formulation. Consider a large language model (LLM) formally defined as $\mathbf{LM} : \mathcal{V}^* \to \mathcal{V}^*$, where $\mathcal{V}$ denotes the vocabulary set and $\mathcal{V}^*$ represents the space of all possible sequences over $\mathcal{V}$. For a given prompt $p \in \mathcal{V}^*$ and input $x \in \mathcal{V}^*$, the LLM processes their concatenation $[p, x] \in \mathcal{V}^*$ to produce an output sequence. Let $\mathcal{D}$ be a distribution over the input-output pairs $(x, y) \in \mathcal{V}^* \times \mathcal{V}^*$. Suppose we have a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, sampled i.i.d from $\mathcal{D}$. The goal of prompt engineering is to identify an optimal prompt $p^* \in \mathcal{V}^*$ that optimizes the model's expected performance on data drawn from a certain distribution. Formally,

$$p^* = \arg\max_{p \in \mathcal{V}^*} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\texttt{Perf} \left(\mathbf{LM}([p, x]), y \right) \right], \tag{1}$$

where $\texttt{Perf} : \mathcal{V}^* \times \mathcal{V}^* \to \mathbb{R}$ is a metric function evaluating the quality of the model's output against the ground truth, and $[p, x]$ denotes the concatenation operation. For simplicity, we do not consider training randomness here.

Automatic Prompt Engineering via Textual Gradient Descent (TGD). Manual prompt engineering requires significant expertise, making it both time-consuming and potentially suboptimal. Automating this process presents unique challenges, primarily due to the discrete nature of the prompt space $\mathcal{V}^*$, which renders traditional gradient-based optimization methods inapplicable. A major line of work in automatic prompt engineering (APE) leverages the capabilities of LLMs themselves to iteratively refine prompts. Earlier approaches directly provide the LLM with previous model predictions alongside ground truth labels, enabling it to refine the current prompt based on observed discrepancies [33, 43]. More recent methods adopt a more interpretable approach, first having language models analyze and summarize errors made with the current prompt, then incorporating these insights for prompt refinements [27, 47, 21]. We present extended related works in Appendix A.

These LLM-based APE techniques are commonly referred to as *Textual Gradient Descent* (TGD) in the literature due to their iterative optimization nature [47]. Here, we argue that it should be defined as *Textual Stochastic Gradient Descent* (TSGD), due to the stochasticity of sampling minibatches of training samples. The prompt p serves as the parameter being optimized. For iteration t, we denote the current prompt as p_t and sample a minibatch of data $\{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^m \sim D$. For x_i, we obtain the model prediction $\widehat{y}_i = \mathbf{LM}([p_t, x_i])$. The prompt update rule is written as:

$$p_{t+1} = \texttt{Update}(\mathbf{LM}, p_t, \{(x_i, y_i, \widehat{y}_i)\}_{i=1}^m)$$

where `Update` is an algorithm that leverages LLMs' capabilities to analyze discrepancies between predictions $\{\widehat{y}_i\}_{i=1}^m$ and ground truth labels $\{y_i\}_{i=1}^m$, yielding an improved prompt p_{t+1}.

Example of Prompt Update Rule. One instantiation of the `Update` algorithm can be formalized as a two-stage process [27, 47]. Given the current prompt p_t and a batch of example triplets

$\{(x_i, y_i, \widehat{y}_i)\}_{i=1}^m$, the `Update` algorithm proceeds as follows:[1][2]

$$\nabla p_t = \mathbf{LM}\big([p_{\text{analyze}},\, p_t,\, \{(x_i, y_i, \widehat{y}_i)\}_{i=1}^m]\big), \quad p_{t+1} = \mathbf{LM}\big([p_{\text{refine}},\, p_t,\, \nabla p_t]\big).$$

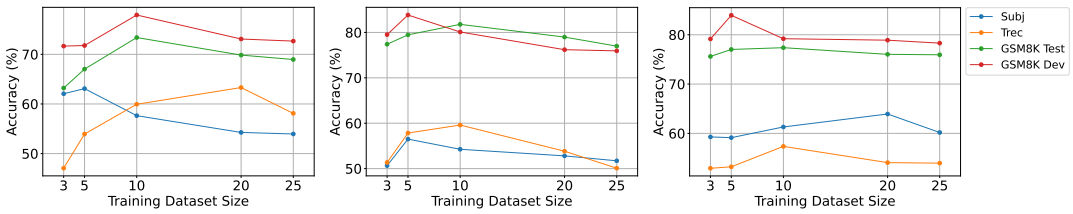
where ∇p_t denotes an error analysis that captures systematic discrepancies between predictions and ground truth, analogous to a gradient in traditional optimization [27]. This textual gradient indicates the direction for prompt improvement, with p_{analyze} directing the LLM to identify error patterns (i.e. creating the gradient) and p_{refine} instructing the LLM to update the prompt accordingly (i.e. performing the *descent* step). This two-stage approach creates a conceptual parallel to how gradient descent uses the negative gradient to move toward optimal parameters iteratively.

3 Test-time Scaling of Data for TGD and TSGD

TGD generates each prompt update using all available examples ("full-batch"), whereas TSGD samples only a subset per iteration, yielding noisier but more scalable textual gradients. Prior work shows that increasing the number of in-context examples can improve accuracy and fluency while reducing variance [10]. However, excessively long contexts may degrade reasoning performance, particularly in single-pass settings [16]. In contrast, TGD and TSGD demand multiple update steps. This raises a key question: how—and how much—should we scale data in these iterative prompt-optimization frameworks? We investigate two complementary scaling studies in greater detail. First, we assess how enlarging the full training corpus affects performance. Second, we isolate the impact of increasing in-context examples under TSGD, holding the underlying training set constant. Following [33], we set the prompt-generation temperature to 0.7 and apply early stopping when holdout-set accuracy fails to improve for 2 iterations. All results are averaged across 10 different runs.

3.1 Test-time Scaling of Training Data for Full-batch TGD

To begin our exploration of data scaling, we first assess how the total size of the training corpus influences performance in the full-batch TGD setting, providing a baseline understanding before we investigate the effects of batch sizing in TSGD. We optimize prompts under TGD frameworks with full batch settings, observing their final downstream task accuracy as the size of training dataset increases. Tasks in TGD are more complicated than common question-answering benchmarks (e.g.,



(a) TextGrad (b) DLN1 (c) DSPy-COPRO

Figure 2: Test performance trends of scaling full training dataset from $N = 3$ to 25 examples.

[16]) in that they demand multi-step reasoning [47] and self-reflection on intermediate failures *per iteration* rather than simple retrieval or labeling. Following the same experimental setups of TextGrad [47], DSPy-COPRO [13] (We direct readers to Appendix A.1 for justification of our choice of COPRO as a baseline.), and DLN1 [33], we evaluate 3 representative datasets under fixed training data and using full batch of training data over the TGD framework: GSM8K (grade-school math problem solving; [5]) (We report GSM8K test and development accuracies under the same settings as [13] and [47], respectively denoted GSM8K Test and GSM8K Dev.), Subj ([24]), and Trec ([38]). For [47] settings, we follow the same setup as their original paper with gpt-4o [22] as **LM** to generate ∇p_t and gpt-3.5-turobo-0125 as inference **LM**. For other methods, we use Llama3-8B [8] for both generating textual gradients and inference **LM**. For each method and dataset, we set

[1] We present a high-level abstraction here, and specific implementations like TextGrad [47] and Learning by Teaching [21] introduce variations such as using different **LM**s for error analysis and prompt refinement.

[2] [33] presents variations like combining $p_{analyze}$ and p_{refine} into a single prompt template but we emphasize the GD nature here, i.e. iteratively improving the prompt based on successes and errors that **LM** made.

the number of demonstrations m per iteration equal to the full training-set size $|D| = N$ such that TGD sees all training examples per iteration. We experiment with number of shots/input contexts, setting $m = N = 3, 5, 10, 20, 25$ training samples. Figure 2 presents the downstream task accuracy $\mathbb{E}_{(x,y)\sim D}\left[\texttt{Perf}\left(\mathbf{LM}([p, x]), y\right)\right]$.

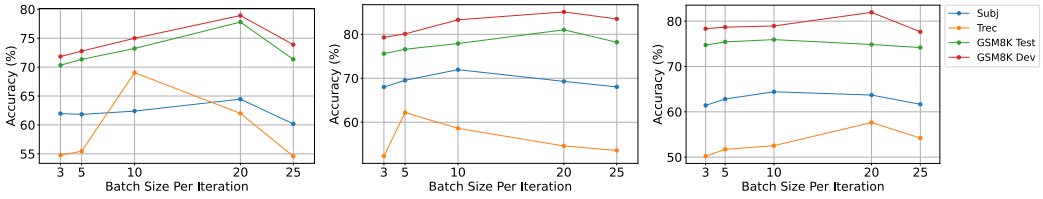
Dataset	Max Tokens (25 ex.)	Avg Tokens per ex.
Subj	≈700	≈28
Trec	≈325	≈13
GSM8K	≈2000	≈80

Table 2: Token statistics with randomly sampled 25 training samples

Findings: In Figure 2, the performance first increases to a clear maximum before declining as more examples are added. However, the optimal "sweet spot" differs across tasks, baselines and models scales: TextGrad often peaks at 10 examples on GSM8K but earlier on Subj. DLN1 generally maximizes between 5 and 10 examples; and DSPy-COPRO peak shifts or even plateaus depending on the dataset. This heterogeneity highlights the challenge of data scaling in APE tasks—without prior knowledge, one cannot *predict the "tipping point"* for training data size that will yield peak performance. We hypothesize that this saturation arises from the degrading reasoning capabilities of increasing input length for current LLMs [16]. To our knowledge, prior work (e.g., [16, 15]) evaluates LLM reasoning over long contexts in a *single-pass* setting, whereas our study *first* investigates the effects of incrementally increasing training data in an *iterative* APE pipeline. Lastly, one might question the sufficiency of the full training data scale to 25 examples as the context length of many state-of-the-art language models shall contain hundreds of samples. We argue that we limit the maximum N to 25 because beyond this point performance declines, and in most real-world tasks—like most tasks in BBH [34]—golden data are scarce. Moreover, [16] systematically evaluated how input length affects reasoning across various models and observed a marked performance drop beyond roughly 500 tokens. To remain consistent for datasets, we therefore limit all in-context samples to 25 for Subj, Trec, and GSM8K (See Table 2).

3.2 Test-time Scaling of Batch Size for TSGD

In this section, we investigate the impact of increasing in-context examples under TSGD while keeping the training set fixed. In this experiment, we hold the training dataset fixed (i.e, N is constant) and vary the per-iteration minibatch size m over $\{3, 5, 10, 20, 25\}$ when estimating textual gradients in TSGD. In optimization literature, we are systematically increasing the stochastic gradient batch size to evaluate its effect on convergence and performance. We repeat the same experimental setting in Section 3.1.



(a) TextGrad (b) DLN1 (c) DSPy-COPRO

Figure 3: Test set performance trends of scaling batch size from $m = 3$ to 25.

Findings: In Figure 3, we find that TextGrad framework overall exhibits larger volatility for different batch sizes due to the next iteration prompt p_{t+1} is one single prompt rather than one prompt chosen out of a set of prompts [33, 13], easily stuck into local minima. In Figure 3 (a), all four tasks (Subj, Trec, GSM8K Test and Dev) exhibit their highest accuracy at moderate batch sizes (10–20 examples), but performance can change sharply outside this range. Notably, Trec performance spikes at batch = 10 (≈ 69 %) yet degrades precipitously at larger batch sizes, and both GSM8K curves peak at batch = 20 before falling back at 25. This "tipping point" behavior underscores a strong sensitivity to batch-size choice. In DLN1 and DSPy-COPRO ((b) (c)), model accuracy rises smoothly as batch size

increases and plateaus over a broader range. Subj and GSM8K Test/Dev all improve steadily to a maximum at batch = 10–20, with only a mild decline at 25; Trec climbs monotonically to its apex at 5 for DLN1 and 20 for DSPy-COPRO. Compared to TextGrad, both DLN1 and DSPy-COPRO show less volatility and higher peak accuracy (e.g. GSM8K Dev $\approx$ 82 %). We highlight that even when the training set size is fixed, the choice of per-update batch size has a dataset-dependent impact on downstream task accuracy, which shall be unpredictable across tasks, baselines and model scales.

4 Textual Stochastic Gradient Descent with Momentum

Previously, we empirically demonstrated that *scaling data is not all we need for TGD and TSGD*. Moreover, we are the first to investigate the scaling effect for iterative workflow without model parameters updates. In this setting, on one hand, estimating textual gradients from mini-batches introduces substantial noise, making it difficult to identify a consistent sweet spot for batch-size scaling. On the other hand, computing full textual gradients is computationally prohibitive and time-consuming, often degrading reasoning capabilities [18].These issues are exacerbated by the limited ability of large language models to reason effectively over long contexts. Inspired by the famous "momentum" technique from neural-network optimization [28, 19], we hope to stabilize noisy textual gradients: the momentum term effectively reduces variances and provides sufficient "inertia" to escape shallow local minima. Here, we show that augmenting TSGD with a momentum term significantly improves task performance and reduces variance compared to TSGD. We also present a theoretical analysis of variance reduction in TSGD-M (see Appendix D). As detailed in Section 2, we maintain a buffer of all past meta-prompts (i.e. the selected prompt p_t for past iterations) and generate each new prompt by adaptively sampling token-by-token from this buffer according to predefined weights until a maximum length is reached. The generated prompt is then either used directly as the next meta-prompt [47] or, in alternate workflows, multiple candidates are produced and the one achieving highest holdout-set accuracy is selected [33, 13] (See Algorithm 1).

Algorithm 1 Textual Stochastic Gradient Descent with Momentum (TSGD-M)

1: **Input: LM**: Language model, p_0: initial prompt, $\mathcal{D}$: data distribution, m: batch size , T: total iterations, use momentum flag, α: momentum parameter, $T_{\max}$: max tokens, k: number of candidate prompts need to generate, S: Score Function, p_{refine}: Template to generate new prompts

2: **for** $t = 0, 1, \ldots, T-1$ **do**

3: Sample batch $\{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^m \sim \mathcal{D}$

4: Compute predictions $\widehat{y}_i^{(t)} = \mathbf{LM}([p_t, x_i^{(t)}])$ for $i = 1, \ldots, m$
▷ Generate next prompt

5: **if** use momentum **then** ▷ Update prompt with momentum

$$p_{t+1} \leftarrow \texttt{Update-Mom}\left(\mathbf{LM}, \alpha, \{p_\tau\}_{\tau=0}^t, \bigcup_{\tau=0}^{t} \{(x_i^{(\tau)}, y_i^{(\tau)}, \widehat{y}_i^{(\tau)})\}_{i=1}^m, T_{max}, k, S, p_{\text{refine}}\right) \text{ (Alg 2)}$$

6: **else** ▷ Update prompt without momentum

$$p_{t+1} \leftarrow \texttt{Update}\left(\mathbf{LM}, p_t, \{(x_i^{(t)}, y_i^{(t)}, \widehat{y}_i^{(t)})\}_{i=1}^m, T_{max}, k, S, p_{\text{refine}}\right) \quad \text{(Alg 3 in Appendix B)}$$

7: **Output:** Optimized prompt p_T

In optimization literature, stochastic gradient descent with momentum (SGDM) shall be written as

$$m^t = \alpha\, m^{t-1} + (1-\alpha)\, \tilde{g}^t, \quad x^{t+1} = x^t - \eta\, m^t.$$

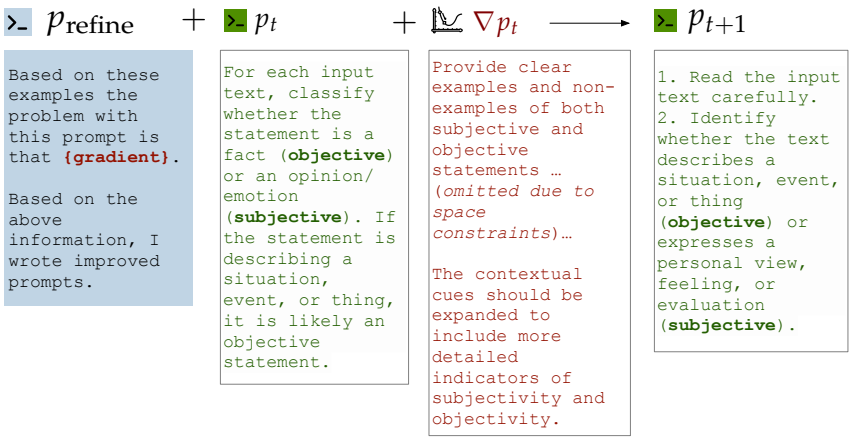
$$m^t = \alpha\, m^{t-1} + (1-\alpha)\, \tilde{\nabla} p_t, \quad p_{t+1} = p_t - m^t\,.$$

in which $\tilde{\nabla} p_t$ denotes the "textual gradient" of current minibatch at current iteration t. We omit the stepsize η here since we perform descent on textual gradient (i.e. using prompt template p_{refine} to generate improved prompt for further iterations). The core of textual gradient descent lies in first generating weights of each past meta prompt $\{p_\tau\}_{\tau=0}^t$ and perform *adaptive* sampling from *mixture of distribution*. We shall introduce the momentum term by storing all the past meta prompts (i.e., the best prompts selected in the previous iterations). We will first generate a list of weights w_τ for

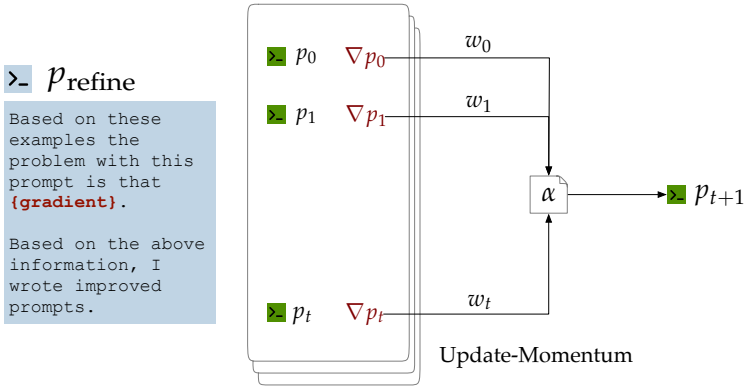
$\tau = 0, 1, 2, ..., t$ (Equation 2) and every weight associated with each meta prompt is generated by exponential moving average, mimicing the spirit of momentum role in SGDM. In our formulation of TGD, we omit the traditional step size η and instead interpret each token generation from the LLM as a sample from a distribution (Equation 2). The term adaptive refers to the way we sample meta-prompts p_i for $i = 0, 1, 2, \ldots, T_{\max}$ and iteratively append previously generated tokens to the selected meta-prompt p_i to generate the next token. This process is adaptive in the sense that the sampling dynamically adjusts based on the current token position.

$$\mathbb{P}(\text{Token}_{i+1} \mid \text{Token}_{1:i}, \{p_\tau\}_{\tau=0}^{t}) \propto \sum_{\tau=0}^{t} w_\tau \mathbb{P}(\text{Token}_{i+1} \mid \text{Token}_{1:i}, p_\tau), \text{ where } w_\tau = \frac{\alpha^{t-\tau}}{\sum_{\tau=0}^{t} \alpha^{t-\tau}} \quad (2)$$

Remark: We consider two cases to introduce momentum term resembling SGDM for Algorithm 1: **Case 1**. The textual gradient descent step is a *single* step, generating the new prompt in one step [33, 13] (See Figure 8 in Appendix C). In this setting, we will perform momentum sampling based on all previous meta prompts p_i per token generation (See Line 7 in Algorithm 2). **Case 2**. The textual gradient descent step separates out the step to generate gradients and step to perform gradient descent [47, 27] (See Figure 4). Following [47] setup, the algorithms would first provide a prompt template



(a) TSGD: new prompts are generated using p_{refine}, current prompt p_t and the "textual gradient" ∇p_t, to guide the LLM in updating the prompt.



(b) Momentum on textual gradients

Figure 4: An illustration of the momentum sampling in Case 2.

p_{analyze} to generate gradients (feedbacks/criticisms) on current prompt p_t, i.e. $\nabla \tilde{p}_t$. Then the gradient is inserted into p_{refine} to refine the current prompt. In this way, we will perform momentum sampling based on both prompt and its corresponding textual gradient $p_i + \nabla p_i$. (See Line 7 in Alg 2).

We also acknowledge the recent work by [47], where *momentum* is defined as the concatenation of past meta-prompts over a fixed window during early iterations. In contrast, our approach defines prompt generation process through momentum sampling from weights applied to each past meta-prompt (Equation 2) and applied per token generation. Our design is notably more memory-efficient: instead

Algorithm 2 `Update-Mom`

Input: **LM**, T_{max}, k, S, p_{refine}, α: momentum parameter, $\{p_\tau\}_{\tau=0}^{t}$: past meta prompts up to iteration t, past batches of sampled demonstrations $\bigcup_{\tau=0}^{t}\{(x_i^{(\tau)}, y_i^{(\tau)}, \hat{y}_i^{(\tau)}\}_{i=1}^{m}$
Generate a list of weights: $\{w_\tau\}_{\tau=0}^{t} = [\frac{\alpha^{t-\tau}}{\sum_\tau^t \alpha^{t-\tau}}]$; Initialize $\tilde{Z} \leftarrow \emptyset$
for $j = 1$ to k **do**
    $\tilde{z} \leftarrow \emptyset$
    for $i = 1$ to T_{max} **do** ▷ Momentum sampling
        Sampling p_i from $\mathbb{P}$, where $\mathbb{P}(p_\tau) = w_\tau$
        Generate one more token t_i using $\mathbf{LM}(p_{\text{refine}} + p_i)$ if using meta prompts (or $\mathbf{LM}(p_{\text{refine}} + p_i + \nabla p_i)$ if using textual gradients) ▷ Token-wise prompt generation
        $\tilde{z} \leftarrow \tilde{z} + [t_i]$
    $\tilde{Z} \leftarrow \tilde{Z} \bigcup \tilde{z}$
if $k = 1$ **then** $p_{t+1} \leftarrow \tilde{Z}$ **else** $p_{t+1} \leftarrow \arg\max_{z \in \tilde{Z}} S(z)$ **end if**
Output: p_{t+1}

of concatenating multiple past meta-prompts, we input a single meta-prompt to generate each new token. Table 3 presents an ablation study comparing vanilla DLN1 against the concatenation-based strategy of [47] for DLN1; a more detailed analysis is provided in Appendix B.

5 Experiments

We conducted a series of experiments to compare the proposed TSGD-M method with TSGD on a range of reasoning, Big-Bench Hard (BBH) and NLU tasks. We find that TSGD-M robustly improves downstream task accuracy for every language models considered, across a range of model scales.

5.1 Experiment Setup

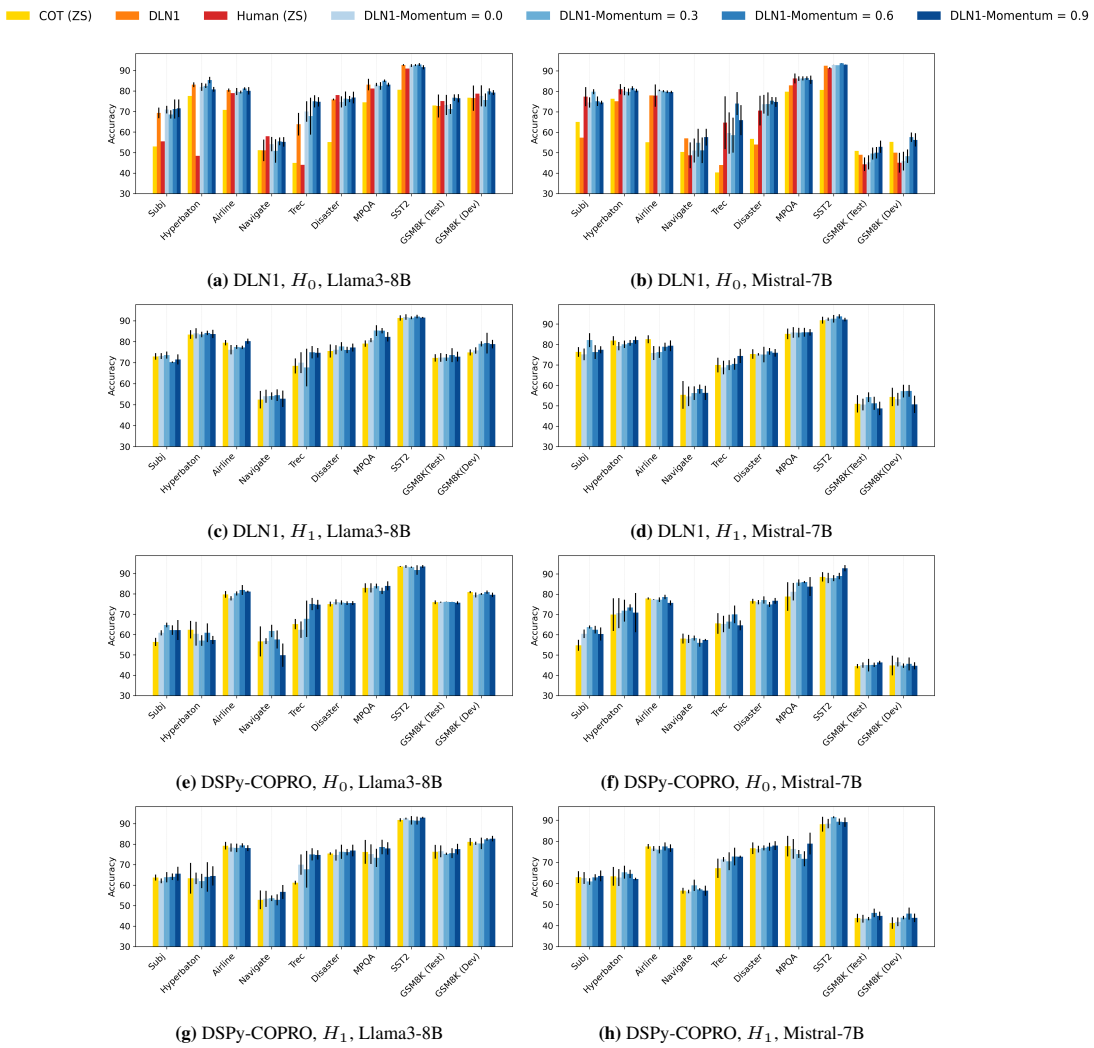
Tasks and Datasets We evaluate TSGD-M on the following benchmarks. We focus on classification tasks. All experiments reported in this section is mean with standard deviation over 10 runs. Full details on the tasks used are given in Appendix E.1.

- **Big Bench Hard (BBH)**: Following [33], we evaluate Navigate (spatial reasoning) and Hyperbaton (adjective ordering) using the 250 BBH-provided examples as the test set.
- **Natural Language Understanding (NLU)**: As in [33], we draw Mpqa, Trec, and Subj from [20], and Disaster and Airline from Leopard [4]. For each dataset, we randomly sample 400 examples for training, 250 for validation, and 250 for testing.
- **Mathematical Reasoning**: Following [47] and [33], we use GSM8K [5] as our dataset and split 200/300/1319 as train/validation/test.

Baselines We evaluate TSGD-M against 3 representative TSGD methods: TextGrad [47], DSPy-COPRO [13], and DLN1 [33]. All experiments are conducted in a zero-shot setting to isolate and demonstrate the impact of our momentum-sampling module within TSGD frameworks. As all above methods are zero-shot prompt tuning methods for APE tasks, we include two human-designed baselines designed for zero shot evaluations:

- **Human (ZS)**: Zero-shot human prompts tailored per dataset. For initial prompts, we use the same initial prompts in [33] and [47] and refer readers on Appendix E.2.
- **COT (ZS)**: "Let's think step by step." for zero-shot chain-of-thought prompt [14].

Language Models For DSPy-COPRO and DLN1, we experiment with Llama3-8B [8], Mistral-7B [12], and DeepSeek-R1-Distill-Qwen-1.5B (denoted as Deepseek 1.5B) [9] with varying model sizes. For TextGrad, we follow [47], using GPT-4o as the gradient generator and GPT-3.5-Turbo as the inference model. Preliminary trials with smaller open-source models showed degrading inference performance (with all three above models)—when the gradient generator was weaker than or equal to the inference model, so we adhere to the original TextGrad experimental settings.



(a) DLN1, H_0, Llama3-8B

(b) DLN1, H_0, Mistral-7B

(c) DLN1, H_1, Llama3-8B

(d) DLN1, H_1, Mistral-7B

(e) DSPy-COPRO, H_0, Llama3-8B

(f) DSPy-COPRO, H_0, Mistral-7B

(g) DSPy-COPRO, H_1, Llama3-8B

(h) DSPy-COPRO, H_1, Mistral-7B

Figure 5: Performance of TSGD-M applied to DLN1 [33], DSPy-COPRO [13] under H_0 and H_1. Under H_0, DLN1-Momentum and DSPy-COPRO-Momentum yield moderate improvements over their vanilla counterparts across most tasks. Under H_1 both momentum-based methods demonstrate more substantial gains, with broader improvements observed across tasks.

Hypotheses We test two hypotheses for TSGD-M against the baselines described above and use them as an ablation study to demonstrate that TSGD-M remains robust across different prompt-generation temperatures and total iterations of prompt refinements.

- H_0 follows the same settings as [33] with a generation temperature of 0.7 and early stopping after 2 iterations no improvement for holdout set accuracy. Under H_0, TSGD-M yields modest improvements over standard TSGD.

- H_1 raises the generation temperature to 1.1 and allows more iterations with early stopping after 5 iterations for no improvement on holdout accuracy, hypothesizing that increased stochasticity and longer optimization will produce more diverse prompts and presumably better downstream performance [46, 40, 29]. Figure 6 confirms that TSGD-M further enhances accuracy compared to H_0 with extended iterations and higher holdout set accuracy

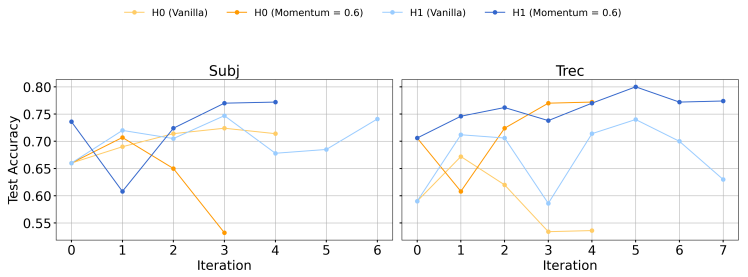


Figure 6: Test Performance comparison of DLN1 vs. DLN1-Momentum$= 0.6$ under H_0 and H_1 on Subj and Trec. Momentum improves test accuracy, especially with extended iterations under H_1.

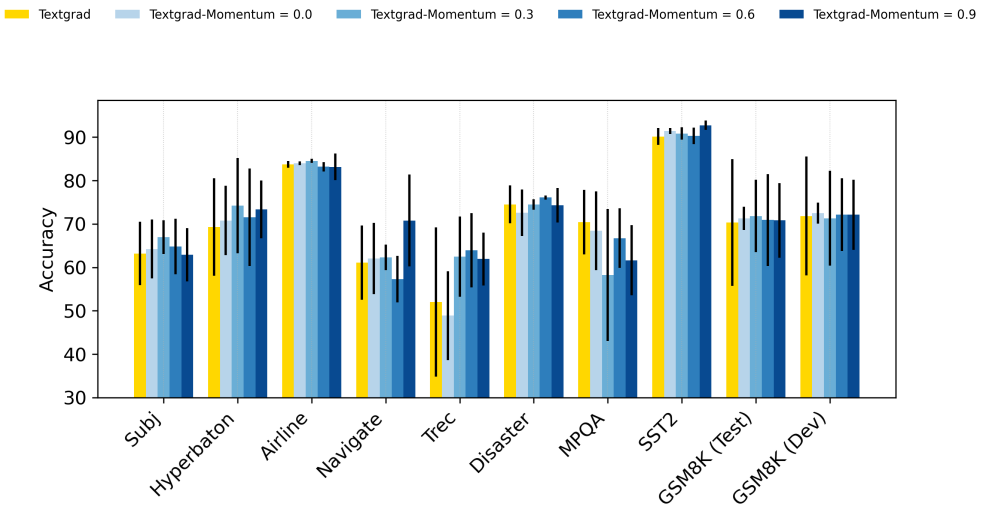


Figure 7: Performance of TextGrad

5.2 Main Results

Figure 5 shows the performance of TSGD-M applied to DLN1 [33] and DSPy-COPRO [13] under both hypotheses H_0 and H_1 with momentum parameter $\alpha \in \{0.0, 0.3, 0.6, 0.9\}$ for Llama3-8B and Mistral-7B. We put results for Deepseek 1.5B in Figure 9 in Appendix F.1. Because H_0 corresponds to the original DLN1 setup—using zero-shot chain-of-thought (CoT (ZS)) and human (ZS) prompts on the same datasets—we only report those two prompt types under H_0, while prompts generated by increased temperature and longer iterations are shown under H_1. Figure 7 presents TSGD-M performance under the original TextGrad protocol. Following [47], we run 12 total iterations with a per-iteration batch size of 3 (36 in-context examples sampled with replacement) and use a prompt-generation temperature of 0 in their implementation.

H_0: We observe that moderate momentum $\alpha = 0.6$ consistently maximizes downstream task accuracy. For Llama3-8B, $\alpha = 0.6$ yields the highest accuracy on 8 of 10 tasks (e.g., Hyperbaton 85.33 % vs. 83.07 % baseline, SST2 92.87 % vs. 92.67 %). Mistral-7B peaks at $\alpha = 0.6$ on most benchmarks (e.g., MPQA 86.50 % vs. 83.13 %, Hyperbaton 81.60 % vs. 81.04 %). For smaller models like Deepseek-1.5B, it also benefits most at $\alpha = 0.6$ (e.g., Subj 65.02 % vs. 61.07 %, Airline 55.87 % vs. 54.23 %). We also find that smaller models like Deepseek 1.5B exhibits larger variance and stronger relative gains of TSGD-M compared to other bigger models like Llama3-8B and Mistral-7B, highlighting the momentum version is more beneficial when the base model is weaker. Lastly, we also observe that $\alpha = 0.9$ often plateaus or slightly degrades accuracy (e.g., Navigate Task with Deepseek 1.5B model drops from 61% at $\alpha = 0.6$ to 57.80 % at $\alpha = 0.9$, confirming that excessive inertia can overshoot local optima.

H_1: Under more aggresive H_1 regime, in which the generation temperature is higher for more diverse prompts and up to 5 iterations of holdout set non-improvement with early stopping, all three models exhibit the largest accuracy gains at moderate momentum ($\alpha = 0.3 - 0.6$). For instance, $\alpha = 0.6$ yields highest lift for most tasks using Llama3-8B and Mistral-7B, while $\alpha = 0.9$ is more suitable for Deepseek-1.5B. This finding validates that a moderate inertia term is optimal for leveraging the increased stochasticity and extended tuning horizon of H_1 echoing [32]'s observation that simply raising the momentum parameter does not improve test-set accuracy in DNN training.

5.3 Additional Studies

TSGD-M is robust to momentum parameter α We test TSGD-M's robustness across three axes. First, sweeping momentum $\alpha \in \{0.0, 0.3, 0.6, 0.9\}$ on 9 tasks under both H_0 and H_1 shows that moderate values ($\alpha = 0.3/0.6$) consistently peak in accuracy, while all momentum variants still outperform vanilla TSGD by 1–3 percentage point. Second, repeating these experiments on three models confirms larger relative gains on smaller models, highlighting momentum's value for weaker language models. Finally, under the original TextGrad protocol (12 iterations, batch size 3, temperature 0), TSGD-M improves over TextGrad, underscoring its stability across hyperparameters and models.

Different notions of momentum In our ablation study, we use DLN1 as a baseline with Llama3-8B as an example language model and compare 2 variants: (1) DLN1 and (2) the concatenation-based momentum strategy of TextGrad [47] (See Table 3). DLN1 Concat prompts perform worse than DLN1 for all datasets and DLN1-Momentum (our version) beats DLN1 for all datasets under the same setting.

Dataset	DLN1	DLN1 Concat
Subj	**69.33(2.57)**	65.22(1.2)
Hyperbaton	**83.07(1.15)**	79.19(1.28)
Airline	**80.40(0.79)**	76.3(0.6)
Navigate	**51.07(5.28)**	44.13(2.57)

Table 3: H_0 : DLN1 vs. DLN1+Concat Prompts (Llama3-8B) for sample datasets.

TSGD-M generates a higher proportion of synthetic examples than vanilla TSGD For DLN1-Momentum = 0.6, the resulting prompts contain substantially more instances recycled from the training set—suggesting that momentum amplifies the model's tendency to "memorize" past error cases and incorporate them into subsequent prompt refinements (See Table 4).

Approach	Optimized Prompt
DLN1	1. Read the question carefully and **identify the main topic**. 2. Determine if the topic is a person, place, thing, or idea. 3. Ask if it refers to a **living being (human/animal), location, concept, object, or event**. 4. Choose the correct category (human, location, entity, description, expression, number). 5. Output the chosen category.
DLN1–Momentum ($\alpha = 0.6$)	1. Read the question carefully and **identify the information type**. 2. Determine whether it asks for: **• A specific quantity (number)** **• A person or group (human)** **• A non-person concept or idea (entity)** **• A descriptive explanation (description)** **• A place or area (location)** 3. Use these guidelines to select and output the correct category.

Table 4: DLN1 vs. DLN1–Momentum ($\alpha = 0.6$) for Trec. Words corresponding to past errors or examples observed during training are highlighted in bold.

Limitations. We would like to note that as our TSGD-M is designed for sampling *per token*, but in reality, we perform momentum generation per every 10 tokens when using API querying for DSPy-COPRO and TextGrad. We refer the readers for reasons in Appendix E.3 for reasons.

6 Conclusion and Discussion

We cast a wide range of automatic prompt engineering workflows as Textual Stochastic Gradient Descent (TSGD), in which mini-batches of examples are sampled at each iteration. Building on this view, we introduce TSGD-Momentum (TSGD-M)—a lightweight plug-in that consistently improves accuracy across BBH, NLU, and reasoning benchmarks. Beyond boosting performance, TSGD-M

stabilizes the optimization trajectory and significantly reduces variance in downstream task accuracy. We also highlight a practical trade-off: the token-wise momentum sampling in TSGD-M adds to the runtime of every iteration. Since longer tuning runs generally yield better accuracy, practitioners may balance iteration cost against performance gains when deploying TSGD-M under time constraints.

References

[1] Rishabh Agarwal, Avi Singh, Lei Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, et al. Many-shot in-context learning. *Advances in Neural Information Processing Systems*, 37:76930–76966, 2024.

[2] Kareem Amin, Alex Bie, Weiwei Kong, Alexey Kurakin, Natalia Ponomareva, Umar Syed, Andreas Terzis, and Sergei Vassilvitskii. Private prediction for large-scale synthetic text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7244–7262, 2024.

[3] Sotiris Anagnostidis and Jannis Bulian. How susceptible are llms to influence in prompts? In *COLM*, 2024.

[4] Trapit Bansal, Rishikesh Jha, and Andrew Mccallum. Learning to few-shot learn across diverse natural language classification tasks. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5108–5123, 2020.

[5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[6] Anthony Cui, Pranav Nandyalam, Ethan Cheung, and Kevin Zhu. Introducing mapo: Momentum-aided gradient descent prompt optimization. *arXiv preprint arXiv:2410.19499*, 2024.

[7] DSPy. Optimization / optimizers. https://dspy.ai/learn/optimization/optimizers/, 2025. Accessed: 2025-05-14.

[8] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[10] Yaru Hao, Yutao Sun, Li Dong, Zhixiong Han, Yuxian Gu, and Furu Wei. Structured prompting: Scaling in-context learning to 1,000 examples. *arXiv preprint arXiv:2212.06713*, 2022.

[11] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference. *arXiv preprint arXiv:2411.09688*, 2024.

[12] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[13] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

[14] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[15] Jinhyuk Lee, Anthony Chen, Zhuyun Dai, Dheeru Dua, Devendra Singh Sachan, Michael Boratko, Yi Luan, Sébastien MR Arnold, Vincent Perot, Siddharth Dalmia, et al. Can long-context language models subsume retrieval, rag, sql, and more? *arXiv preprint arXiv:2406.13121*, 2024.

[16] Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353, 2024.

[17] Linyu Liu, Yu Pan, Xiaocheng Li, and Guanting Chen. Uncertainty estimation and quantification for llms: A simple supervised approach. *arXiv preprint arXiv:2404.15993*, 2024.

[18] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[19] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33:18261–18271, 2020.

[20] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, 2022.

[21] Xuefei Ning, Zifu Wang, Shiyao Li, Zinan Lin, Peiran Yao, Tianyu Fu, Matthew Blaschko, Guohao Dai, Huazhong Yang, and Yu Wang. Can llms learn by teaching for better reasoning? a preliminary study. *Advances in Neural Information Processing Systems*, 37:71188–71239, 2024.

[22] OpenAI. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/, May 2024. Accessed: 2025-05-13.

[23] Krista Opsahl-Ong, Michael Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, 2024.

[24] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 271–278, Barcelona, Spain, July 2004.

[25] Hao Peng, Xiaozhi Wang, Jianhui Chen, Weikai Li, Yunjia Qi, Zimu Wang, Zhili Wu, Kaisheng Zeng, Bin Xu, Lei Hou, et al. When does in-context learning fall short and why? a study on specification-heavy tasks. *arXiv preprint arXiv:2311.08993*, 2023.

[26] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[27] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

[28] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[29] Matthew Renze. The effect of sampling temperature on problem solving in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, 2024.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[31] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*, 2024.

[32] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[33] Alessandro Sordoni, Eric Yuan, Marc-Alexandre Côté, Matheus Pereira, Adam Trischler, Ziang Xiao, Arian Hosseini, Friederike Niedtner, and Nicolas Le Roux. Joint prompt optimization of stacked llms using variational inference. *Advances in Neural Information Processing Systems*, 36:58128–58151, 2023.

[34] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[35] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[36] Xinyu Tang, Richard Shin, Huseyin A Inan, Andre Manoel, Fatemehsadat Mireshghallah, Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, and Robert Sim. Privacy-preserving in-context learning with differentially private few-shot generation. In *The Twelfth International Conference on Learning Representations*, 2024.

[37] Together AI. Chat api overview – long-running conversations. https://docs.together.ai/docs/chat-overview, 2025. Accessed April 2025.

[38] Ellen M Voorhees and Dawn M Tice. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207, 2000.

[39] Xingchen Wan, Ruoxi Sun, Hootan Nakhost, and Sercan Arik. Teach better or show smarter? on instructions and exemplars in automatic prompt optimization. *Advances in Neural Information Processing Systems*, 37:58174–58244, 2024.

[40] Chi Wang, Xueqing Liu, and Ahmed Hassan Awadallah. Cost-effective hyperparameter optimization for large language model generation inference. In *International Conference on Automated Machine Learning*, pages 21–1. PMLR, 2023.

[41] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

[42] Chulin Xie, Zinan Lin, Arturs Backurs, Sivakanth Gopi, Da Yu, Huseyin A Inan, Harsha Nori, Haotian Jiang, Huishuai Zhang, Yin Tat Lee, et al. Differentially private synthetic data via foundation model apis 2: Text. *arXiv preprint arXiv:2403.01749*, 2024.

[43] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

[44] Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, et al. Internlm-math: Open math large language models toward verifiable reasoning. *arXiv preprint arXiv:2402.06332*, 2024.

[45] Xinjie Yu and Mitsuo Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

[46] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in Neural Information Processing Systems*, 36:55734–55784, 2023.

[47] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[48] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2022.

A Extended Related Works

A.1 Automatic Prompt Engineering Workflow

We will revisit several **Automatic Prompt Engineering** frameworks below.

1. **APE** [48] is a seminal work in leveraging LLMs for instruction optimization. In each iteration, a set of instructions is evaluated on a validation set, and the optimizer generates a new set by paraphrasing the highest-performing instructions. This iterative process continues until convergence. However, we argue that APE does not fall under the category of Textual Gradient Descent (TGD) but instead aligns more closely with evolutionary algorithms [45], as it is inherently gradient-free. Rather than utilizing textual gradients for optimization, APE explicitly prompts LLMs to generate variations of instructions while preserving their semantic meaning, replacing lower-performing prompts through mechanisms akin to random variation (e.g., mutation or crossover), a hallmark of evolutionary strategies. Therefore, we exclude it for our evaluation.
2. **DLN1** [33] views prompt optmization as learning a distribution $p_{LM}(y|x, \pi)$ in which x, y are inputs or outputs separately, and π is learnable prompt. The iterative process is similar to APE but can include a verbalization of difficult examples from the task: the final prompts shall combine both instructions and task examples, which mimic a mix of zero-shot learning and in-context learning.
3. **OPRO** [43] optimizes instructions by presenting the trajectory of previously generated prompts with their corresponding training set accuracy, together with randomly extracted demonstrations from the training set to denote the task of interest. The algorithm only keep instructions with highest scores in the meta-prompt in consideration of LLM context length limit. The iterative process only asks LLMs to generate one more new prompt per iteration. Note here [43] typically runs much longer iterations compared to DLN1 [33] and DLN1 shall serve as a shorter version of [43]. We argue that due to its similarity to DLN1, we use DLN1 as a representative method to evaluate our TSGD-M algorithm.
4. **TextGrad** [47] backpropogates textual feedbacks provided by the proposal and view the textual feedbacks as gradients to perform descent or improve upon. For every iteration, they randomly extract several demonstrations and generate only one new prompt. They also present a momentum version by simply concatenating previously generated past gradients within certain window length.
5. **DSPy** [13]. As we limit our study into zero-shot prompt optimization, in which we solely focus on *instruction optimization* rather than *example optimization* or jointly optimize both of them [39, 23], we only discuss COPRO module in [13]. As our tasks are APE with zero-shot demonstrations needed to optimize, we use COPRO for automatic instruction optimization and exclude MIPROv2 as our baselines do not involve optimizing the set of few shots demonstrations. Similar to DLN-1, COPRO leverages Signatures (structured prompts) to optimize Signatures themselves. We refer readers for further discussions on different optimizers [7].
6. **PromptAgent** [41] views prompt optimization as a more advanced planning agent using Monte Carlo Tree Search (MCTS). We argue that [41] does not fall under TGD framework also. The MCTS algorithm itself is not a gradient based algorithm as it relies on a search based approach rather than differentiable optimization techniques and MCTS does not compute or apply gradients. Even though MCTS shall be combined with gradient base learning where a policy network is trained using policy gradients and used to guide tree search, it is beyond our paper's research scope. Thus, we exclude this method.
7. **ProTeGi** [27] was among the first methods to incorporate gradient descent principles into automatic prompt generation. Our TSGD-M framework can be naturally extended to **ProTeGi**. Specifically, we propose performing token-wise sampling over batches of *meta-prompts*. Rather than applying momentum sampling to individual meta-prompts from p_0 to p_{t-1}, we instead sample across batches of prompts, denoted by $\cup_{\tau=1}^{i} B_\tau$. From this union, we select a batch B_i and apply uniform weights to all prompts within that batch. We exclude this method for evaluation due to double sampling but this method shall be viewed as further research direction. We are also aware of a concurrent line of work on momentum integration in ProTeGi [6], where a history of past gradients is maintained and *a single gradient is randomly sampled to generate a new prompt pool at each iteration*. In contrast, our approach performs adaptive sampling with decayed weights (defined by momentum parameter) over past gradients (or meta-prompts) at the token level, continuing until the maximum token limit is reached.

A.2 Synthetic Text Generation

Recent work in differentially private (DP) language model training has explored synthetic generation as a mechanism to protect sensitive data while enabling downstream utility [36, 2, 42]. Notably, prior approaches such as those in DP-Few-Shot Generation [36] construct synthetic datasets by prompting an LLM to generate tokens one at a time, with differential privacy applied via logit-level mechanisms such as clip-and-aggregate, Gaussian noise, or report-noisy-max [2]. These methods often rely on carefully controlled sampling from private logits or fallback to public logits when logit similarity allows, in order to minimize privacy cost. In contrast, our approach focuses on momentum-based prompt synthesis, where token-by-token generation is guided not by privacy constraints, but by a trajectory of previously optimized prompts—analogous to a gradient descent path in prompt space. While our framework does not aim for differential privacy, it shares structural similarities with the above methods in generating synthetic text autoregressively under external constraints (e.g., past prompt trajectories). This connection highlights the broader utility of iterative prompt conditioning in synthetic data pipelines, whether for privacy-preserving learning or for optimizing instruction-following behavior via momentum sampling.

B Additional algorithm details

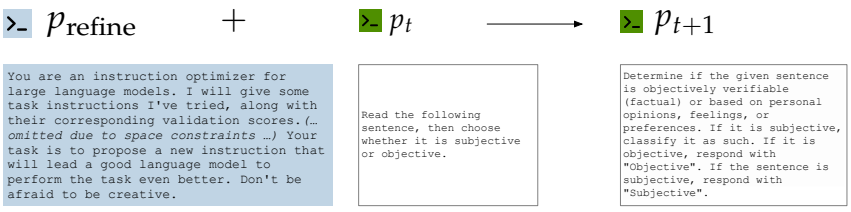
Below Algorithm 3 provides psuedo-code for vanilla TSGD per iteration.

Algorithm 3 `Update`

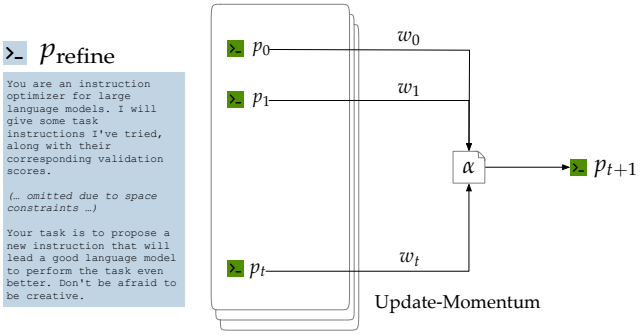
Input: Language model **LM**, current prompt p_t, $\{(x_i^{(t)}, y_i^{(t)}, \widehat{y}_i^{(t)})\}_{i=1}^m$, max tokens T_{max}, number of candidate prompts need to generate k, Score Function S, LLM Template to generate new prompts p_{refine}
$\tilde{Z} \leftarrow \emptyset$
for $j = 1$ to k **do**
    $\tilde{z} \leftarrow \emptyset$
    for $i = 1$ to T_{max} **do**
        Generate one more token t_i using **LM**$(p_{\text{refine}} + p_t)$
        $\tilde{z} \leftarrow \tilde{z} + [t_i]$
    $\tilde{Z} \leftarrow \tilde{Z} \bigcup \tilde{z}$
if $k = 1$ **then**
    $p_{t+1} \leftarrow \tilde{Z}$
else
    $p_{t+1} \leftarrow \arg\max_{z \in \tilde{Z}} S(z)$
Output: p_{t+1}

C Illustration of Momentum Sampling (Case 1)

Figure 8 illustrates the momentum sampling process described in Section 4 for Case 1, where textual gradient descent operates in a single step [33, 13]. In this example, the refinement prompt p_{refine} is instantiated using a textual example from the DSPy-COPRO framework and is concatenated with the current meta prompt p_t to generate the next round of candidate prompts.



(a) TSGD: new prompts are generated using p_{refine} prompt template to instruct the LLM in updating the prompt p_t meta prompt from last iteration.



(b) Momentum on meta-prompts

Figure 8: An illustration of the momentum sampling in Case 1.

D Theoretical Justification

Setting. We consider a simplified setting where the optimal prompt is a scalar $\mu \in \mathbb{R}$. In each iteration, the LLM samples from $\mu+\epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is independent noise. Prompt performance is measured by mean squared error (MSE) $\mathbb{E}[(p-\mu)^2]$.

Let the baseline approach generate prompts $\{X_t\}$ where:

$$X_t = \text{LLM} \tag{3}$$

and the alternative approach (momentum) generate prompts $\{Y_t\}$ using exponential moving average

$$Y_t = \alpha \cdot \text{LLM} + (1-\alpha) \cdot Y_{t-1}, \quad 0 < \alpha < 1 \tag{4}$$

for all $t \geq 1$. We note that $Y_0 = \text{LLM}$.

Theorem 1 (Variance Reduction due to Exponential Moving Average)**.** *Then for all $t \geq 1$, we have $\mathbb{E}[X_t] = \mathbb{E}[Y_t] = \mu$, and*

$$\mathbb{E}[(X_t - \mu)^2] = \mathbb{E}[\epsilon_t^2] = \sigma^2 \tag{5}$$

and

$$\begin{aligned}\mathbb{E}[(Y_t - \mu)^2] &= \alpha^2 \sum_{k=0}^{t-1} (1-\alpha)^{2k} \sigma^2 \\ &= \sigma^2 \left[\frac{\alpha}{2-\alpha} + \frac{2}{2-\alpha}(1-\alpha)^{2t+1} \right]\end{aligned}$$

Therefore, the momentum approach achieves strictly lower MSE.

Proof. The baseline process follows $X_t = \mu + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. At iteration t:

$$\mathbb{E}[(X_t - \mu)^2] = \mathbb{E}[\epsilon_t^2] = \sigma^2 \tag{6}$$

The momentum approach can be expanded recursively as follows:

$$Y_t - \mu = (1-\alpha)^t \epsilon_0 + \alpha \sum_{k=1}^{t} (1-\alpha)^{t-k} \epsilon_k$$

where $\epsilon_k \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$. We can easily see that $\mathbb{E}[Y_t - \mu] = 0$. Hence, the mean squared error becomes

$$\begin{aligned}\mathbb{E}[(Y_t - \mu)^2] = \text{Var}(Y_t - \mu) &= (1-\alpha)^{2t}\sigma^2 + \alpha^2 \sum_{k=1}^{t} (1-\alpha)^{2(t-k)} \sigma^2 \\ &= \sigma^2 \left[(1-\alpha)^{2t} + \alpha^2 \frac{1-(1-\alpha)^{2t}}{1-(1-\alpha)^2} \right] \\ &= \sigma^2 \left[(1-\alpha)^{2t} + \frac{\alpha}{2-\alpha}(1-(1-\alpha)^{2t}) \right] \\ &= \sigma^2 \left[\frac{\alpha}{2-\alpha} + \frac{2}{2-\alpha}(1-\alpha)^{2t+1} \right] \\ &< \sigma^2\end{aligned}$$

and as $t \to \infty$, we have

$$\mathbb{E}[(Y_t - \mu)^2] \to \frac{\alpha}{2-\alpha}\sigma^2$$

□

E More Experiment Details

E.1 Task Description

In Table 5, we provide brief descriptions and dataset statistics for all tasks used in our experiments. For GSM8K [5], we adopt the same data split as [13, 47], using 200 examples for training, 300 for validation/development, and 1319 for testing.

Task	|train|	|valid|	|test|	|class|	Description
Mpqa	400	256	250	2	Sentiment analysis.
Trec	400	256	250	6	Question type classification.
Subj	400	256	250	2	Determine whether a sentence is subjective or objective.
Disaster	400	250	250	2	Determine whether a sentence is relevant to a disaster.
Airline	400	250	250	3	Airline tweet sentiment analysis.
Hyperbaton	400	1000	250	2	Order adjectives correctly in English sentences.
Navigate	375	375	250	2	Spatial reasoning given navigation instructions.
SST2	67349	872	872	2	Sentiment analysis.
GSM8K	200	300	1319	N/A	Reasoning Task.

Table 5: Tasks used in this work.

E.2 Prompt Initialization

We initialize the task description as reported in Table 6 for all tasks and evaluations. For GSM8K[5], we use the default system prompt [13] as initialization.

Task	Initialization
Mpqa	Read the following review, then choose whether it is negative or positive.
Trec	Read the following question, then choose whether it is about a description, entity, expression, human, location or number.
Subj	Classify the input text as subjective or objective.
Disaster	Read the following sentence, then choose whether it is relevant to a disaster.
Airline	Read the following sentence, then choose whether it is positive, negative, or neutral.
Hyperbaton	Which sentence has the correct adjective order.
Navigate	Read the following sentence, then determine whether you return to the starting point.
SST2	Classify the input text as positive or negative.
GSM8K	Your input fields are: 1. ‘question‘ (str) Your output fields are: 1. ‘reasoning‘ (str) 2. ‘answer‘ (str) All interactions will be structured in the following way, with the appropriate values filled in. [[## question ##]] question [[## reasoning ##]] reasoning [[## answer ##]] answer [[## completed ##]] In adhering to this structure, your objective is: Given the fields ‘question‘, produce the fields ‘answer‘.

Table 6: Prompt initializations.

E.3 Templates

For DLN1, we adopt the same forward classification template as used in [33] for computing predictions of mini-batches (forward pass). For DSPy-COPRO [13] and TextGrad [47], we follow the prompt templates provided in their respective papers.

Classification Template Forward Pass

{{ prompt }}

{{ input }}

Answer:

General API Templates As an example of an API template, we provide the following configuration for DSPy_COPRO, which uses the Together AI platform [37] as the API provider and Llama-3 8B [8] as the selected language model.

Together API Chat Example for DSPy-COPRO

import os
from together import Together
client = Together()
response = client.chat.completions.create(
model="together_ai/meta-llama/Meta-Llama-3-8B-Instruct-Turbo",
messages=[{"role": "system", "content": "Your input fields are:
1.attempted_instructions (str)
Your output fields are:
1. ‘proposed_instruction‘ (str): The improved instructions for the language model
2. ‘proposed_prefix_for_output_field‘ (str): The string at the end of the prompt, which will help the model start solving the task.
All interactions will be structured in the following way, with the appropriate values filled in.
[[## attempted_instructions ##]]
{attempted_instructions} [[## proposed_instruction ##]] {proposed_instruction }
[[## proposed_prefix_for_output_field ##]]
proposed_prefix_for_output_field
[[## completed ##]]
In adhering to this structure, your objective is:
You are an instruction optimizer for large language models. I will give some task instructions I’ve tried, along with their corresponding validation scores. The instructions are arranged in increasing order based on their scores, where higher scores indicate better quality.
Your task is to propose a new instruction that will lead a good language model to perform the task even better. Don’t be afraid to be creative." },
"role": "user", "content": [##attempted_instructions##]
[1] Instruction #1 : Analyze the sentiment of the given sentence by considering the tone, language, and context, ...
[2] Prefix #1 : The sentiment of the sentence is: [3] Resulting Score #1 : 40.0
[4] Instruction #2 : Analyze the given sentence carefully, considering the context, tone, and language used to express the sentiment. Evaluate the emotional undertones, such as excitement, sadness, or frustration, to determine the overall sentiment of the sentence. Classify the sentiment as positive if it expresses happiness, satisfaction, or approval, negative if it expresses dissatisfaction, anger, or sadness, and neutral if it states a fact or shows no emotional tone. [5] Prefix #2 : The sentiment of the given sentence is:
[6] Resulting Score #2 : 43.3... },
{" "role": "assistant", "content": "Analyze the sentiment of the given sentence by considering the language, } ▷ **Comment: We concatenate all previously generated tokens here.**
],
"temperature": 0.7, "api_key": XXX, "n": 1, "max_tokens": 10 ▷ **Comment: We choose 10 as max new tokens generated for every momentum sampling round.**
)
print(response.choices[0].message.content)

In our experiments, we observed that many platforms recommended by DSPy [13] and TextGrad [47] do not support token-wise generation at the granularity of every single token. For example, the OpenAI platform does not allow true token-by-token generation. Similarly, the Together API also lacks support for generating tokens one at a time. We experimentally found that if using deepseek models, together AI API platform would output "<think ><think ><think >" consecutively; if using Llama models, together AI API platform would output "[[[[[[" based on the above template as our template explicitly asks model to output [1] Instruction. As a practical workaround, we perform momentum sampling every 10 tokens (refer to as "max_tokens": 10), which we found to be effective in practice. Specifically, we find that extending the generation length to 10 tokens reduces token-level repetition observed in single-token generation, while still preserving the benefits of momentum sampling.

E.4 Implementation Details

Same as [33], for all tasks, we set max tokens to be generated $T_{\max} = 100$ for all tasks except GSM8K. For DLN1 and DSPy-COPRO, the number of candidate prompts need to generate $k = 20$ and batch size $m = 20$ for all tasks. We set the total iterations T is 20 while most of our iterations end in 10 as we set the early stopping.

E.5 Runtime

For both TextGrad and DSPy-COPRO experiments, across all datasets, the total runtime remains under 1 hour for both the momentum and non-momentum variants when executed on CPU. For DLN1, we report runtime using LLaMA3-8B as a representative model, using 4 NVIDIA A40 GPUs.

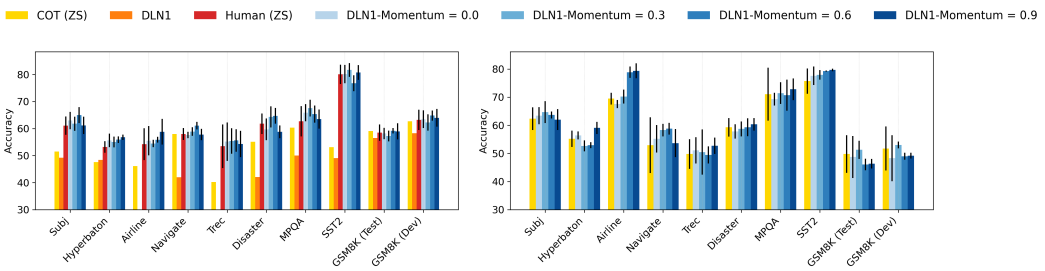
Table 7: Llama3-8B: Averaged Runtime across 10 trials (in hours) of DLN1 and DLN1-Momentum across datasets under H_0 (no prompt optimization) and H_1 (with prompt optimization).

Dataset	**DLN1** (H_0)	**DLN1-M** (H_0)	**DLN1** (H_1)	**DLN1-M** (H_1)
Subj	1	1.5	2	4
Hyperbaton	4	5	2	3.5
Airline	0.3	0.5	0.5	2.5
Navigate	0.1	1	0.2	4
Trec	1	2.5	0.5	4
Disaster	0.1	0.6	0.3	3
MPQA	0.2	0.5	1.2	3
SST2	1	2	1.5	3
GSM8K	3	8	6	7.5

F More Experiments

F.1 Results for Deepseek 1.5B Models

Figure 9 is all the results for DLN1 and DSPy-COPRO under H_0 and H_1 for Deepseek 1.5B.



(a) DLN1, H_0, Deepseek 1.5B

(b) DLN1, H_1, Deepseek 1.5B

(c) DSPy-COPRO, H_0, Deepseek 1.5B

(d) DSPy-COPRO, H_1, Deepseek 1.5B

Figure 9: Performance of TSGD-M applied to DLN1 [33], DSPy-COPRO [13] under H_0 and H_1 for Deepseek 1.5B.

F.2 Result Tables for Figure 5

Below we provide the original statistics for Figure 5 in Section 5 with various α values. The reported statistics is averaged across 10 trials with standard deviation in parentheses. We provide DLN1 and DSPy-COPRO under both hypotheses for H_0 and H_1. Under H_0, we report only the results of CoT (ZS) and Human (ZS) for DLN1, as these are the only methods that do not involve any form of prompt optimization or iterative generation. All other reported methods operate in an iterative manner and incorporate prompt refinement or optimization strategies. Both CoT (ZS) and Human (ZS) are evaluated on the test set using greedy decoding (temperature $= 0$), therefore no standard deviation for these two methods. DLN1 and DSPy-COPRO use the same dataset setup so we put COT (ZS) and Human (ZS) under DLN1. We emphasize that incorporating momentum into existing modules reduces variance for both DLN1 and DSPy-COPRO. Similar to SGD with momentum, where a typical setting like $\alpha = 0.9$ is commonly used but not universally optimal, TSGD-M achieves variance reduction across a range of $\alpha > 0$ values (though dataset dependent), without requiring fine-tuning for peak performance. Table 20 denotes the momentum addition compared to vanilla Textgrad [47] under the original settings with gpt-3.5-turbo-0125 (inferencing for p_{refine}) and gpt-4o (generating gradients for $p_{analyze}$). For TextGrad, we repeat each experiment over 10 random seeds and find that the improved test/development accuracy trajectory is not consistently reproducible (especially for GSM8K). Interestingly, for tasks such as Subj, Hyperbaton, SST2, and GSM8K, the prompts obtained without the TextGrad optimization framework sometimes outperform those generated through the iterative process. Among the configurations tested, a momentum value of $\alpha = 0.3$ tends to yield performance that ranks among the top two across most datasets.

Dataset	COT(ZS)	DLN1	Human(ZS)	DLN1 -Momentum = 0.0	DLN1-Momentum = 0.3	DLN1-Momentum = 0.6	DLN1-Momentum = 0.9
Subj	53.0	69.03(2.57)	55.5	70.93(1.83)	68.57(2.01)	71.20(4.63)	**71.57(3.29)**
Hyperbaton	77.6	83.07(1.15)	48.4	82.00(1.96)	82.53(1.10)	**85.53(1.53)**	80.8(1.18)
Airline	70.8	80.20(0.79)	79.0	79.73(1.72)	79.47(0.66)	**81.47(0.62)**	80.10(1.79)
Navigate	51.20	51.07(5.28)	**58.0**	54.13(3.46)	50.80(5.72)	55.47(1.86)	55.20(2.29)
Trec	45.00	63.80(5.49)	44.0	70.00(4.98)	67.73(8.96)	**75.00(2.97)**	74.73(2.29)
Disaster	55.10	75.83(0.42)	**78.0**	74.67(2.71)	76.27(3.47)	76.07(1.53)	76.83(2.85)
MPQA	74.45	83.13(2.81)	81.2	83.13(0.79)	82.47(1.85)	**85.02(0.58)**	83.12(0.88)
SST2	80.6	92.67(0.37)	90.94	92.37(0.70)	92.60(0.50)	**92.87(0.69)**	91.57(1.02)
GSM8K (Test)	72.90	72.67(5.56)	75.12	73.2(4.89)	71.2(2.4)	**76.75(1.6)**	76.45(2.0)
GSM8K (Development)	76.70	76.53(6.16)	78.73	77.60(5.12)	75.6(3.2)	**79.80(1.5)**	79.23(1.2)

Table 8: DLN1: H_0 Hypothesis for Lllama3-8B

Dataset	COT(ZS)	Human(ZS)	DLN1	DLN1 - Momentum = 0.0	DLN1 - Momentum = 0.3	DLN1 - Momentum = 0.6	DLN1 - Momentum = 0.9
Subj	65.10	57.35	77.4(4.67)	74.6(2.50)	**79.87(1.21)**	75.2(2.32)	74.6(0.85)
Hyperbaton	76.40	75.2	81.04(2.52)	80.13(2.05)	79.73(1.71)	**81.6(0.95)**	80.4(0.85)
Airline	55.10	78.00	77.97(5.45)	**80.5(0.36)**	80.1(0.54)	79.80(0.64)	79.67(0.58)
Navigate	50.40	57.0	48.72(6.29)	51.08(5.89)	54.88(6.82)	51.20(6.23)	**57.6(4.07)**
Trec	40.40	44.0	64.70(12.9)	59.6(10.1)	58.6(8.52)	**74.07(5.62)**	65.9(7.34)
Disaster	56.8	54.0	70.60(7.26)	73.73(4.56)	73.8(5.67)	**75.45(1.81)**	74.8(2.4)
MPQA	79.85	83.0	86.3(2.41)	86.23(1.11)	86.34(1.00)	**86.5(0.86)**	85.67(2.05)
SST2	80.73	92.55	91.37(0.45)	92.9(0.04)	92.8(0.03)	**93.8(0.02)**	93.1(0.03)
GSM8K (Test)	50.9	49.0	44.37(3.34)	45.29(3.45)	49.6(2.9)	50.0(2.84)	**52.9(2.98)**
GSM8K (Development)	55.3	50.0	45.1(4.83)	45.91(4.6)	48.3(3.3)	**57.6(2.3)**	56.3(3.2)

Table 9: DLN1: H_0 Hypothesis for Mistral-7B

Dataset	COT(ZS)	Human(ZS)	DLN1	DLN1 - Momentum = 0.0	DLN1 - Momentum = 0.3	DLN1 - Momentum = 0.6	DLN1 - Momentum = 0.9
Subj	51.50	49.25	61.07(3.41)	62.98(3.20)	61.78(2.65)	**65.0(2.04)**	61.19(3.22)
Hyperbaton	47.60	48.4	53.2(2.14)	55.6(2.50)	55.04 (1.99)	55.89(1.22)	**56.8(0.99)**
Airline	46.10	18.1	54.23(5.90)	55.5(5.40)	54.47(1.35)	55.87(1.07)	**58.83(4.75)**
Navigate	58.00	42.0	58.0(2.23)	57.64(1.17)	58.9(1.58)	**61.07(1.41)**	57.8(2.1)
Trec	40.20	27.6	53.48(8.07)	55.15(7.10)	55.4(4.72)	**55.6(4.17)**	54.3(4.89)
Disaster	55.10	42.1	61.78(3.82)	59.71(4.02)	64.32(3.89)	**64.7(2.9)**	58.80(2.34)
MPQA	60.35	50.00	62.70(5.59)	65.87(3.15)	**67.58(3.10)**	65.35(3.18)	63.50(3.54)
SST2	53.10	49.08	80.07(3.58)	80.17(3.38)	**81.67(2.53)**	76.77(2.98)	80.77(2.73)
GSM8K (Test)	59.12	56.50	58.53(3.02)	58.20(2.1)	57.23(1.98)	**59.2(1.02)**	58.96(2.99)
GSM8K (Development)	62.7	58.3	63.23(3.77)	63.46(3.29)	62.26(3.04)	**64.82(1.89)**	63.96(3.28)

Table 10: DLN1: H_0 Hypothesis for Deepseek 1.5B

Dataset	DSPy-COPRO	DSPy-COPRO Momentum = 0.0	DSPy-COPRO Momentum = 0.3	DSPy-COPRO Momentum = 0.6	DSPy-COPRO Momentum = 0.9
Subj	56.37(1.96)	60.86(1.27)	**64.70(1.10)**	62.2(2.36)	62.23(4.92)
Hyperbaton	**62.40(4.33)**	60.40(5.89)	57.07(2.60)	60.93(4.60)	57.33(2.01)
Airline	79.73(1.68)	77.8(1.02)	80.2(1.04)	**81.9(2.42)**	81.13(0.45)
Navigate	56.67(7.38)	56.80(1.44)	**61.73(3.03)**	57.60(4.45)	49.87(5.75)
Trec	65.13(2.54)	62.40(4.02)	67.73(8.96)	**75.00(2.97)**	74.73(2.29)
Disaster	74.97(1.15)	**76.03(1.22)**	75.77(1.04)	75.43(0.90)	75.47(0.94)
MPQA	82.93(2.29)	82.93(2.29)	83.83(1.16)	81.47(1.57)	**83.93(2.21)**
SST2	**93.47(0.25)**	93.43(0.68)	93.10(0.35)	91.70(2.34)	93.40(0.70)
GSM8K (Test)	75.86(0.87)	75.90(0.29)	**76.07(0.05)**	76.02(0.04)	75.6(0.71)
GSM8K (Development)	80.86(0.51)	79.43(1.22)	79.97(0.47)	**80.87(0.80)**	79.53(1.08)

Table 11: DSPy-COPRO: H_0 Hypothesis for Lllama3-8B

Dataset	DSPy-COPRO	DSPy-COPRO - Momentum = 0.0	DSPy-COPRO - Momentum = 0.3	DSPy-COPRO - Momentum = 0.6	DSPy-COPRO - Momentum = 0.9
Subj	54.80(2.70)	60.47(2.10)	**63.83(0.72)**	62.50(1.87)	60.40(3.21)
Hyperbaton	69.96(8.07)	70.65(7.40)	71.88(5.37)	**73.45(1.56)**	70.93(9.64)
Airline	77.90(0.56)	77.40(0.17)	77.30(1.15)	**78.63(1.07)**	75.73(1.25)
Navigate	58.13(2.46)	57.93(1.99)	**58.40(1.20)**	56.07(1.93)	57.47(0.46)
Trec	65.61(5.1)	65.33(4.05)	66.45(3.42)	**70.05(4.38)**	64.68(2.46)
Disaster	76.57(1.16)	76.07(1.01)	**77.13(1.79)**	74.93(1.21)	76.73(1.42)
MPQA	78.87(7.05)	81.23(4.30)	85.70(1.65)	**86.00(0.53)**	83.77(4.65)
SST2	88.57(2.35)	88.0(2.52)	87.93(1.46)	88.93(1.57)	**92.83(1.52)**
GSM8K (Test)	44.53(1.01)	45.07(1.28)	45.03(3.11)	**45.17(1.11)**	**46.33(0.81)**
GSM8K (Development)	44.87(4.83)	**46.57(2.23)**	44.77(1.08)	45.63(3.21)	44.70(1.73)

Table 12: DSPy-COPRO: H_0 Hypothesis for Mistral-7B

Dataset	DSPy-COPRO	DSPy-COPRO - Momentum = 0.0	DSPy-COPRO - Momentum = 0.3	DSPy-COPRO - Momentum = 0.6	DSPy-COPRO - Momentum = 0.9
Subj	53.23(2.5)	54.53(2.37)	**56.90(1.25)**	56.37(1.44)	55.77(2.24)
Hyperbaton	47.0(3.62)	46.23(2.98)	45.97(1.63)	49.0(2.60)	**50.10(2.64)**
Airline	55.23(3.30)	56.33(3.27)	57.19(2.91)	**57.23(0.21)**	55.91(2.98)
Navigate	70.67(2.54)	69.77(1.05)	71.60(1.39)	71.97(3.03)	**72.67(2.31)**
Trec	48.2(3.63)	46.00(1.97)	49.48(3.35)	49.93(5.31)	**51.28(3.20)**
Disaster	50.1(2.89)	49.03(1.18)	50.20(2.81)	50.50(1.28)	**50.97(1.07)**
MPQA	66.57(3.89)	64.23(2.92)	68.70(3.39)	**69.17(3.19)**	67.88(3.41)
SST2	72.1(2.96)	70.87(1.42)	71.0(6.15)	72.47(2.80)	**72.87(2.91)**
GSM8K (Test)	78.47(3.21)	78.43(0.12)	**79.97(0.21)**	79.70(0.92)	79.77(0.38)
GSM8K (Development)	82.47(0.88)	82.77(1.50)	83.00(1.18)	**83.47(0.68)**	81.33(3.45)

Table 13: DSPy-COPRO: H_0 Hypothesis for Deepseek 1.5B

Hypothesis 1

Dataset	DLN1	DLN1 - Momentum = 0.0	DLN1 - Momentum = 0.3	DLN1 - Momentum = 0.6	DLN1 -Momentum = 0.9
Subj	72.97(1.56)	73.2(1.39)	**73.67(1.68)**	70.25(0.32)	71.57(2.37)
Hyperbaton	83.44(2.07)	84.0(2.47)	83.46(1.32)	**84.2(1.07)**	83.67(1.98)
Airline	79.50(1.27)	76.2(2.24)	77.5(0.94)	77.3(0.78)	**80.27(1.21)**
Navigate	52.40(4.17)	54.00(3.12)	54.1(1.78)	**54.53(2.78)**	52.80(3.89)
Trec	68.47(3.55)	70.00(4.98)	67.73(8.96)	**75.00(2.97)**	74.73(2.29)
Disaster	75.58(3.12)	76.16(2.17)	**77.73(2.02)**	76.07(1.53)	77.23(1.88)
MPQA	79.12(1.54)	80.73(0.98)	**85.32(2.59)**	85.27(1.28)	82.35(2.20)
SST2	91.32(1.27)	91.77(1.30)	91.40(0.70)	**91.97(0.84)**	91.53(0.34)
GSM8K(Test)	72.3(1.74)	72.5(2.05)	72.5(1.63)	**73.6(3.33)**	72.9(2.32)
GSM8K(Development)	74.90(1.40)	75.80(1.52)	79.0(1.21)	**79.33(4.93)**	78.9(2.02)

Table 14: DLN1: H_1 Hypothesis for Llama3-8B

Dataset	DLN1	DLN1 - Momentum = 0.0	DLN1 - Momentum = 0.3	DLN1 - Momentum = 0.6	DLN1 - Momentum = 0.9
Subj	76.33(2.40)	75.15(2.88)	**82.13(3.38)**	76.3(3.37)	77.4(1.71)
Hyperbaton	81.87(2.24)	79.2(2.04)	80.10(1.82)	80.8(1.39)	**82.13(1.79)**
Airline	**82.57(1.92)**	75.83(3.43)	76.27(2.82)	78.87(2.00)	79.37(2.66)
Navigate	55.33(6.8)	54.6(4.8)	56.2(3.34)	**58.2(2.21)**	56.3(3.45)
Trec	69.93(3.59)	68.73(3.34)	69.90(2.42)	70.46(2.84)	**74.4(3.46)**
Disaster	75.3(2.34)	75.17(0.60)	75.08(3.81)	**76.63(1.66)**	75.90(1.87)
MPQA	85.2(2.58)	85.87(2.59)	85.78(2.32)	**85.97(2.18)**	85.93(1.62)
SST2	91.87(1.70)	92.40(0.78)	92.60(1.85)	**93.77(1.04)**	92.17(0.87)
GSM8K(Test)	50.93(4.25)	50.56(2.85)	**54.2(2.37)**	51.2(3.2)	48.67(3.27)
GSM8K(Development)	54.33(4.47)	53.2(3.14)	57.2(3.12)	**57.3(2.9)**	50.67(4.25)

Table 15: DLN1:H_1 Hypothesis for Mistral-7B

Dataset	DLN1	DLN1 - Momentum = 0.0	DLN1 - Momentum = 0.3	DLN1 - Momentum = 0.6	DLN1 - Momentum = 0.9
Subj	62.37(4.03)	63.42(3.03)	**64.67(3.9)**	63.67(1.25)	62.0(3.72)
Hyperbaton	55.20(2.88)	56.40(1.39)	52.67(1.97)	52.93(1.08)	**59.06(2.17)**
Airline	69.47(2.1)	67.57(1.46)	70.2(2.47)	78.87(2.00)	**79.37(2.66)**
Navigate	52.93(9.91)	55.20(4.87)	58.30(2.23)	**58.77(2.10)**	53.67(5.00)
Trec	49.8(5.26)	51.12(4.67)	50.48(8.03)	49.48(3.03)	**52.70(2.90)**
Disaster	59.27(3.32)	57.83(2.54)	58.70(2.65)	59.3(3.16)	**60.33(2.21)**
MPQA	71.07(9.43)	69.3(2.27)	71.43(3.85)	70.70(5.52)	**72.8(3.84)**
SST2	75.70(4.52)	77.6(3.32)	77.93(1.69)	79.27(0.2)	**79.67(0.47)**
GSM8K (Test)	49.78(6.71)	48.72(7.45)	**51.28(3.21)**	46.07(2.08)	46.33(1.68)
GSM8K (Development)	51.73(7.82)	48.3(8.2)	**53.0(1.21)**	49.0(1.21)	49.2(1.02)

Table 16: DLN1: H_1 Hypothesis for Deepseek 1.5B

Dataset	DSPy-COPRO	DSPy-COPRO Momentum = 0.0	DSPy-COPRO Momentum = 0.3	DSPy-COPRO Momentum = 0.6	DSPy-COPRO Momentum = 0.9
Subj	63.7(1.47)	62.10(1.17)	63.87(2.49)	64.07(1.59)	**65.60(3.35)**
Hyperbaton	63.33(7.49)	63.27(2.81)	62.0(3.49)	64.0(7.21)	**64.5(4.68)**
Airline	79.23(1.86)	78.33(1.99)	78.17(2.06)	**79.37(0.99)**	78.03(1.42)
Navigate	52.8(4.57)	53.32(3.92)	53.60(1.42)	52.81(2.67)	**56.67(3.40)**
Trec	61.2(0.91)	70.00(4.98)	67.73(8.96)	**75.00(2.97)**	74.73(2.29)
Disaster	75.4(0.57)	74.67(2.71)	76.27(3.47)	76.07(1.53)	**76.83(2.85)**
MPQA	76.20(5.81)	75.10(4.82)	73.30(4.43)	**78.57(3.54)**	77.93(3.01)
SST2	91.8(0.80)	92.47(0.46)	91.60(2.04)	91.47(1.98)	**92.90(0.44)**
GSM8K (Test)	76.27(3.37)	76.50(2.90)	75.27(0.55)	75.57(2.40)	**77.53(2.57)**
GSM8K (Development)	81.13(1.82)	80.39(0.8)	80.33(2.81)	82.33(0.65)	**82.67(1.35)**

Table 17: DSPy-COPRO: H_1 Hypothesis for Lllama3-8B

Dataset	DSPy-COPRO	DSPy-COPRO - Momentum = 0.0	DSPy-COPRO - Momentum = 0.3	DSPy-COPRO - Momentum = 0.6	DSPy-COPRO - Momentum = 0.9
Subj	62.97(2.82)	62.53(2.81)	60.93(1.55)	62.90(1.47)	**63.57(2.55)**
Hyperbaton	63.33(4.44)	62.77(4.03)	**65.33(3.05)**	64.50(1.98)	62.10(0.57)
Airline	77.57(1.14)	76.53(1.15)	76.03(1.79)	**77.53(1.94)**	76.73(1.81)
Navigate	56.53(1.36)	56.13(0.83)	**59.07(2.66)**	57.20(0.57)	56.53(2.37)
Trec	67.2(4.55)	71.43(1.10)	70.53(4.27)	**72.77(4.21)**	72.63(0.57)
Disaster	76.70(2.73)	76.27(1.56)	76.83(1.18)	77.43(1.87)	**77.93(2.09)**
MPQA	77.73(4.86)	76.40(4.68)	73.97(1.89)	71.70(3.56)	**78.90(5.15)**
SST2	88.07(3.52)	88.37(2.23)	**91.43(0.46)**	89.27(1.50)	89.07(2.22)
GSM8K (Test)	43.6(2.02)	43.23(1.86)	43.27(0.81)	**46.00(2.04)**	44.57(2.04)
GSM8K (Development)	41.2(2.71)	41.77(2.04)	43.77(0.81)	**45.67(2.87)**	43.67(2.03)

Table 18: DSPy-COPRO: H_1 Hypothesis for Mistral-7B

Dataset	DSPy-COPRO	DSPy-COPRO - Momentum = 0.0	DSPy-COPRO-Momentum = 0.3	DSPy-COPRO-Momentum = 0.6	DSPy-COPRO-Momentum = 0.9
Subj	53.23(2.49)	53.03(0.76)	53.93(2.37)	55.30(1.23)	**56.17(1.07)**
Hyperbaton	48.67(4.18)	49.47(3.49)	50.87(0.96)	**51.42(1.08)**	50.06(2.17)
Airline	55.4(2.97)	56.40(2.32)	57.10(1.89)	55.80(2.34)	**58.33(1.52)**
Navigate	72.30(4.71)	73.01(3.51)	**73.60(2.83)**	72.90(2.81)	73.20(1.89)
Trec	43.53(4.90)	46.60(4.65)	46.27(3.13)	47.20(3.28)	**48.50(3.98)**
Disaster	54.46(7.23)	54.07(3.32)	55.43(3.63)	**59.4(4.89)**	55.9(5.89)
MPQA	63.66(8.57)	61.93(6.61)	62.8(7.78)	65.38(5.52)	**70.27(3.18)**
SST2	68.33(3.92)	67.30(3.19)	**72.30(3.21)**	69.33(3.23)	69.97(2.15)
GSM8K (Test)	75.52(3.29)	75.67(2.61)	76.17(3.17)	75.13(3.20)	**77.20(1.97)**
GSM8K (Development)	80.87(3.74)	79.93(2.43)	80.90(3.13)	**82.35(2.59)**	81.20(3.20)

Table 19: DSPy-COPRO: H_1 Hypothesis for Deepseek 1.5B

Textgrad

Dataset	COT(ZS)	Human(ZS)	Textgrad	Textgrad - Momentum = 0.0	Textgrad - Momentum = 0.3	Textgrad - Momentum = 0.6	Textgrad - Momentum = 0.9
Subj	**70.45**	64.1	63.20(7.28)	64.23(6.76)	66.99(3.89)	64.80(6.41)	62.92(6.12)
Hyperbaton	85.2	**88.0**	69.28(11.22)	70.80(7.97)	74.26(10.97)	71.52(11.25)	73.36(6.67)
Airline	84.30	83.30	83.70(0.78)	84.0(0.42)	**84.53(0.47)**	83.17(1.11)	83.15(3.10)
Navigate	48.8	37.6	61.10(8.57)	62.04(8.23)	62.30(2.95)	57.28(5.35)	**70.80(10.61)**
Trec	53.40	45.40	52.0(17.21)	48.9(10.24)	62.45(9.23)	**63.91(8.55)**	61.92(6.12)
Disaster	73.8	69.0	74.48(4.36)	72.58(5.35)	74.50(1.22)	**76.10(0.46)**	74.28(3.97)
MPQA	59.10	51.44	**70.42(7.39)**	68.43(9.06)	58.22(15.19)	66.73(6.89)	61.64(8.06)
SST2	91.17	**93.12**	90.14(1.95)	91.40(0.69)	90.84(1.41)	90.27(1.89)	92.72(1.08)
GSM8K (Test)	67.25	**79.31**	70.32(14.63)	71.27(2.71)	71.82(8.32)	70.92(10.59)	70.83(8.58)
GSM8K (Development)	70.0	**82.0**	71.83(13.72)	72.48(2.45)	71.32(10.94)	72.12(8.37)	72.11(8.11)

Table 20: Textgrad: gpt-3.5-turbo-0125(inferencing for p_{refine}) + gpt-4o(gradient for $p_{analyze}$)

F.3 Various Momentum Interpretations

As [47] noted that an alternative interpretation of momentum is earlier iterations of the gradient variable when making the *descent* step. Specifically, [47] simply concatenating all the textual gradients together by the prompt template below.

TextGrad Momentum prompt

Here are the past iterations of this variable:
`<PAST_ITERATIONS>{past_values}</PAST_ITERATIONS>`

To investigate the role of momentum in TextGrad [47], we compared our DLN1 approach with a baseline that mimics the momentum strategy used in TextGrad—specifically, concatenating all past textual gradients (i.e., meta-prompts) (See $B_{\pi}^{ConcatPrompts}$ below), which adapts from Appendix $D.2$ in [33]). We present results for both Llama3-8B (See Table 21, and Table 22) [8], Mistral-7B (See Table 23 and Table 24) [12] and DeepSeek-R1-Distill-Qwen-1.5B (See Table 25 and Table 26) [9].

This concatenation-based strategy was tested under both H_0 and H_1 prompt selection conditions. Across both settings for Llama3-8B, DLN1 consistently outperformed the concatenation method on a wide range of tasks. For instance, under H_0 setting, DLN1 achieved significantly better performance in Subj (69.33 vs. 66.22), Airline (80.40 vs. 78.3), and Disaster (75.83 vs. 74.33), among others. Under H_1, the advantages of DLN1 were even more pronounced, such as on Hyperbaton (83.44 vs. 70.67) and MPQA (79.12 vs. 78.40). These results demonstrate that simply concatenating all previous gradient prompts, as done in TextGrad, is not an effective mechanism for leveraging momentum in APE workflow. By contrast, vanilla DLN1 setting leads to consistently stronger performance across diverse language understanding tasks. Note here we did not list our methods of momentum performance here as most of our methods of momentum outperform vanilla DLN1.

Under the H_1 hypothesis using the Mistral-7B model, DLN1 consistently outperformed the concatenated meta-prompt baseline across a wide variety of datasets. This supports our intuition that simply appending all past prompts—as done in the momentum interpretation of TextGrad—can degrade performance, particularly as the sequence becomes longer and noisier. DLN1 led to stronger results on core language understanding tasks such as Airline (82.57 vs. 64.7), Trec (69.93 vs. 47.07), and MPQA (85.2 vs. 69.37), showing substantial margins of improvement. Notably, DLN1 also demonstrated better performance on reasoning-heavy datasets like GSM8K (Test) (50.93 vs. 48.26) and GSM8K (Dev) (54.33 vs. 52.2), even though the differences were subtler. These results reinforce our claim that raw accumulation of past meta prompts yields less effective generalization and task performance, particularly when scaling up to stronger models like Mistral-7B.

With the DeepSeek-R1-Distill-Qwen-1.5B model [9] —representing the smallest model in our evaluation suite—we observed that DLN1 still outperformed the concatenation-based baseline across most datasets, under both H_0 and H_1 hypotheses. Notably, the performance gains with DLN1 were accompanied by higher variance compared to larger models like Mistral-7B and LLaMA3-8B, which is expected given the reduced capacity and stability of smaller language models. For instance, under H_1 hypothesis , DLN1 surpassed the concatenation method on challenging tasks such as Airline (69.47 vs. 53.43), MPQA (71.07 vs. 59.68), and GSM8K (Dev) (51.73 vs. 49.2), but also exhibited substantial standard deviations (e.g., MPQA ±9.43, Disaster ±7.54). These results further substantiate that prompt concatenation, as used in TextGrad-style momentum, fails to scale down effectively to lower-capacity models. DLN1's performance, while more variable, remains more robust and reliable, suggesting that even for compact models, simply concatenating past meta prompts or accumulating past gradients would bring in performance degradation, which aligns with other research findings that LLMs might not be able to effectively perform summarizations for long context tasks [18, 25].

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**69.33(2.57)**	66.22(1.2)
Hyperbaton	**83.07(1.15)**	81.19(2.28)
Airline	**80.40(0.79)**	78.3(0.6)
Navigate	**51.07(5.28)**	50.13(8.57)
Trec	**63.80(5.49)**	59.0(7.44)
Disaster	**75.83(0.42)**	74.33(5.63)
MPQA	**83.13(2.81)**	80.40(2.82)
SST2	**92.67(0.37)**	92.13(1.31)
GSM8K (Test)	**72.67(5.56)**	71.62(3.34)
GSM8K (Development)	**76.53(6.16)**	74.23(7.21)

Table 21: Llama3-8B: H_0 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**72.97(1.56)**	66.55(3.90)
Hyperbaton	**83.44(2.07)**	70.67(4.78)
Airline	**79.50(1.27)**	71.8(0.64)
Navigate	**52.40(4.17)**	50.03(4.08)
Trec	68.47(3.55)	**72.47(3.73)**
Disaster	**75.58(3.12)**	68.83(3.76)
MPQA	**79.12(1.54)**	78.40(2.82)
SST2	**91.32(1.27)**	78.06(7.04)
GSM8K (Test)	**72.3(1.74)**	70.82(2.31)
GSM8K (Development)	**74.90(1.40)**	71.3(3.56)

Table 22: Llama3-8B: H_1 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**77.4(4.67)**	66.75(11.01)
Hyperbaton	**81.04(2.52)**	76.0(2.80)
Airline	**77.97(5.45)**	75.23(7.51)
Navigate	48.72(6.29)	**56.53(1.40)**
Trec	**64.70(12.9)**	57.07(5.47)
Disaster	**74.97(1.15)**	59.33(8.03)
MPQA	**86.3(2.41)**	73.73(9.59)
SST2	**91.37(0.45)**	90.33(5.51)
GSM8K (Test)	**44.37(3.34)**	42.26(2.7)
GSM8K (Development)	**45.1(4.83)**	43.2(2.3)

Table 23: Mistral-7B: H_0 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**76.33(2.40)**	75.67(6.66)
Hyperbaton	**81.87(2.24)**	75.23(1.26)
Airline	**82.57(1.92)**	64.7(3.33)
Navigate	**55.33(6.8)**	47.67(9.81)
Trec	**69.93(3.59)**	47.07(0.02)
Disaster	**75.3(2.34)**	57.37(0.91)
MPQA	**85.2(2.58)**	69.37(5.65)
SST2	**91.87(1.70)**	90.33(5.51)
GSM8K (Test)	**50.93(4.25)**	48.26(2.7)
GSM8K (Development)	**54.33(4.47)**	52.2(2.3)

Table 24: Mistral-7B: H_1 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**61.07(3.41)**	54.67(2.63)
Hyperbaton	**53.2(2.14)**	48.47(0.41)
Airline	54.23(5.90)	**61.3(2.48)**
Navigate	**58.0(2.23)**	58.0(3.23)
Trec	**53.48(8.07)**	39.7(5.25)
Disaster	**61.78(3.82)**	56.97(1.01)
MPQA	**62.70(5.59)**	58.33(3.41)
SST2	**80.07(3.58)**	72.33(3.10)
GSM8K (Test)	**60.53(1.88)**	58.53(2.3)
GSM8K (Development)	**63.23(2.77)**	60.2(3.3)

Table 25: DeepSeek-R1-Distill-Qwen-1.5B: H_0 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Dataset Name	DLN1	DLN1 Concat Prompts
Subj	**62.37(4.03)**	54.83(2.48)
Hyperbaton	**55.20(2.88)**	53.27(3.83)
Airline	**69.47(2.1)**	53.43(2.58)
Navigate	52.93(9.91)	**56.67(0.19)**
Trec	**49.8(5.26)**	43.8(5.62)
Disaster	**59.27(3.32)**	55.87(7.54)
MPQA	**71.07(9.43)**	59.68(4.67)
SST2	**75.70(4.52)**	72.33(3.10)
GSM8K (Test)	**49.78(6.71)**	47.53(5.32)
GSM8K (Development)	**51.73(7.82)**	49.2(4.83)

Table 26: DeepSeek-R1-Distill-Qwen-1.5B: H_1 hypothesis for DLN1 vs. DLN1 Concatenating Meta Prompts

Prompt Proposal Template $B_{\pi}^{ConcatPrompts}$ for DLN1 Concat Prompts

template:

```
A student is completing a task that requires producing a text output from a text
input. The student receives an instruction that describes how to produce the output
given each input.
The student has made some errors. Your task is to improve the instruction such that
the student can fix the errors.

Here are the past iterations of the instruction.
## Instructions
> {{ past_prompts }}
[END] ▷ Comment: We concatenate all past meta prompts together here.

# Student successes
{% for backward_info in backward_infos %} {% if backward_info.loss == 0.0 %}
## Input:
> {{ backward_info.input }}
## Correct Output:
> {{ backward_info.target }}
{% endif %} {% endfor %}

# Student errors
{% for backward_info in backward_infos %} {% if backward_info.loss > 0.0 %}
## Input:
> {{ backward_info.input }}
## Student Output:
> {{ backward_info.output }}
## Correct Output:
> {{ backward_info.target }}
{% endif %} {% endfor %}

Improve the instruction to fix the student errors. {{ message }}
## Instruction
>
```

message_alternatives:

- Clarify the instruction by adding few words or a short sentence. Be concise.
- Improve the instruction by providing examples on how to solve the task. Be concise.
- Shorten the instruction by removing superfluous words or sentences.
- Rewrite the instruction by providing detailed information to avoid ambiguity. Be concise.

F.4 Optimized Prompts by Momentum Based Sampling

We use DLN1 as a representative method and Llama3-8B as a representative model to illustrate the impact of our momentum-based tokenwise sampling approach. The prompts generated using this strategy consistently incorporate more detailed explanations and illustrative guidance compared to those produced by non-momentum baselines. In this section, we present optimized prompts across all tasks, highlighting how momentum-based tokenwise sampling yields qualitatively different prompts from both human-written and original DLN1-generated prompts. We restrict the number of maximum tokens to be generated by 100. For GSM8k task, we restrict max generated tokens to be 500 and the starting prompt is the default systematic prompt provided by DSPy [13]. Since the hyperparameter for momentum parameter $\alpha = 0.6$ seems to outperform other hyperparameters settings for DLN1-Momentum, we only present prompts for DLN1-Momentum=0.6. For simplicity, we only present prompts generated under H_0 hypothesis.

Approach	Optimized Prompt	Acc.
Human	Classify the input text as subjective or objective.	0.491
DLN1	1. Carefully read the input text. 2. Identify the type of language used in the text. 3. Determine if the text includes words that express the author's opinion, emotion, or perspective. Look for words such as "I", "me", "my", "we", "us", "our", "believe", "think", "feel", "opinion", "perspective", "view", "attitude", "emotion", etc. If it does...	0.713
DLN1-Momentum = 0.6	Classify each input text as subjective or objective. Subjective texts express a personal opinion, emotion, or experience.They often use words and phrases like: - "I think", "I believe", "I feel", "my opinion", "my experience", "I love", "I hate", etc. - Use of first-person pronouns (I, me, my) - Words that describe emotions (e.g., beautiful, sad, excited) - Use of evaluative language (e.g "smart and alert , thirteen conversations about one thing is a small gem .	0.770

Table 27: Prompt comparison for the **Subj** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance. DLN1-Momentum improves the prompt by keeping the task description after iterative refinement.

Approach	Optimized Prompt	Acc.
Human	Which sentence has the correct adjective order.	0.750
DLN1	1. Read the sentence carefully. 2. Identify the adjective(s) in the sentence. 3. Arrange the adjectives in the correct order: descriptive adjective (if any), size adjective, color adjective, shape adjective, material adjective, and finally the noun. 4. Check if the adjective order is correct for each sentence. This improved instruction should help the student to identify and fix the errors in the adjective order. The student can follow the steps and learn to produce the correct output given each input.	0.816
DLN1-Momentum = 0.6	1. Follow the correct order of adjectives: opinion, size, shape, material, color, and origin. 2. Start with an opinion adjective if it is present. If not, start with a size adjective. 3. Then, add a shape adjective, followed by a material adjective, a color adjective, and finally, an origin adjective. 4. If an adjective is missing, leave a space for it. 5. Remember, each adjective should be separated by a space.	0.872

Table 28: Prompt comparison for the **Hyperbaton** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis.

Approach	Optimized Prompt	Acc.
Human	Read the following sentence, then choose whether it is positive, negative, or neutral.	0.701
DLN1	1. Read the sentence carefully. 2. Identify the tone of the sentence. 3. Determine if the tone is positive, negative, or neutral. 4. Write the correct output (positive, negative, or neutral) based on your analysis. Additional Tips: - Pay attention to the words used in the sentence, such as "thanks", "I love", "good", "bad", "happy", "sad", etc. - Look for phrases that indicate a problem or issue.	0.802
DLN1-Momentum = 0.6	1. Read the tweet carefully and identify the key phrases, words, or sentences that convey the sentiment. 2. Determine if the tweet is expressing a positive, negative, or neutral sentiment. 3. Consider the tone and language used in the tweet. 4. Identify the correct output by classifying the sentiment as positive, negative, or neutral. 5. Remember, even if the tweet is mentioning a specific airline, the sentiment expressed may not necessarily be about the airline itself.	0.817

Table 29: Prompt comparison for the **Airline** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. DLN1-Momentum improves the prompt quality by inferring the training examples are tweets rather than sentence.

Approach	Optimized Prompt	Acc.
Human	Read the following question, then choose whether it is about a description, entity, expression, human, location or number.	0.451
DLN1	1. Read the question carefully and identify the main topic. 2. Determine if the topic is a person, place, thing, or idea. 3. Ask yourself if the topic is a living thing (human, animal, or plant), a location, a concept, an object, or an event. 4. Choose the correct category (human, location, entity, description, expression, or number) from the options. 5. Write the chosen category as the output.	0.586
DLN1-Momentum = 0.6	1. Read the question carefully and identify the type of information being asked. 2. Determine whether the question is seeking: * A specific quantity (number) * A person's name or a group of people (human) * A concept, expression, or idea that is not a person (entity) * A description or explanation of something (description) * A location or geographical area (location) 3. Use the guidelines to categorize the correct output type for each problem.	0.772

Table 30: Prompt comparison for the **Trec** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. DLN1-Momentum improves the prompt by presenting a description for each type of answer for Term Clarification

.

Approach	Optimized Prompt	Acc.
Human	Read the following sentence, then choose whether it is relevant to a disaster.	0.621
DLN1	Read the following sentence carefully and decide whether it is relevant to a disaster. Look for information in the sentence that indicates a potential catastrophe or serious harm to people, property, or the environment. If the sentence is about a disaster or could be related to one, answer "yes". Otherwise, answer "no". Remember, the focus is on disasters and the potential harm they can cause.	0.725
DLN1-Momentum = 0.6	Read the sentence carefully and determine if it's relevant to a disaster or crisis, such as a natural disaster, accident, or serious event. Ignore any non-disaster related information, such as personal opinions, jokes, or advertisements. Consider only the main topic or event mentioned in the sentence. If the sentence is about a disaster or crisis, answer "yes", otherwise, answer "no".	0.807

Table 31: Prompt comparison for the **Disaster** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. DLN1-Momentum improves the prompt by providing longer Solution Guidance and more detailed Term Clarification on disaster descriptions.

Approach	Optimized Prompt	Acc.
Human	Read the following review, then choose whether it is negative or positive.	0.785
DLN1	Read the review and identify whether it is a negative or positive review based on the language used. Consider the tone and the words used in the review. Determine if the review uses words that convey a sense of disapproval, criticism, or negativity. If the review uses words that convey a sense of approval, praise, or positivity, then the review is positive. Conversely, if the review uses words that convey a sense of disappointment, frustration, or dissatisfaction, then the review is negative.	0.810
DLN1-Momentum = 0.6	1. Read the review carefully and identify the words or phrases that convey a positive or negative sentiment. 2. Consider the meaning of the words or phrases in the context of the review. 3. Determine if the overall tone of the review is positive or negative. 4. If you're unsure, look for specific words or phrases that clearly indicate a positive or negative sentiment. 5. Write your answer as 'positive' or 'negative'.6. Make sure to consider the context.	0.856

Table 32: Prompt comparison for the **Mpqa** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. DLN1-Momentum generates prompts with assertive tone Emphasis on answering the question.

Approach	Optimized Prompt	Acc.
Human	Classify the input text as positive or negative.	0.840
DLN1	1. Read the input text carefully. 2. Determine the overall tone of the text. 3. Classify the tone as positive, negative, or neutral. If the text expresses a positive sentiment, such as praise, admiration, or enthusiasm, classify it as positive. If the text expresses a negative sentiment, such as criticism, dislike, or frustration, classify it as negative. If the text does not express a clear sentiment, classify it as neutral. 4. Write the corresponding output.	0.922
DLN1-Momentum = 0.6	1. Read the input text carefully and identify the main idea or sentiment expressed. 2. Determine whether the text expresses admiration, criticism, or a neutral opinion. If it expresses admiration, write "positive". If it expresses criticism, write "negative". If it is neutral, write "neutral". 3. Be cautious not to misinterpret the text, especially when it uses sarcasm or irony. 4. Practice reading between the lines and considering the tone and context of the text.	0.935

Table 33: Prompt comparison for the **SST2** task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. DLN1-Momentum generates prompts with more direct and accurate Solution Guidance

.

Approach	Optimized Prompt	Acc.
Human	Your input fields are: 1. ‘question‘ (str) Your output fields are: 1. ‘reasoning‘ (str) 2. ‘answer‘ (str) All interactions will be structured in the following way, with the appropriate values filled in. [[## question ##]] question [[## reasoning ##]] reasoning [[## answer ##]] answer [[## completed ##]] In adhering to this structure, your objective is: Given the fields ‘question‘, produce the fields ‘answer‘.	0.6
DLN1	Your task is to write a text output given a text input. The input will be in the format of a question, and you need to produce a reasoning and an answer. The structure of the output is: [[## question ##]] question [[## reasoning ##]] reasoning [[## answer ##]] answer To complete this task, you will need to follow these steps: 1. Read the input question carefully. 2. Understand what is being asked and what information is needed to solve the problem. 3. Write a clear and concise reasoning for your answer. Make sure your reasoning is logical and easy to follow. 4. Calculate the answer based on the given information. 5. Write the answer in the format of a sentence or a number. Remember to use proper mathematical operations and follow the given instructions carefully. If you are unsure about any step, you can try breaking it down into smaller parts or using a calculator to help you with the calculation. Also, double-check your answer to ensure it is reasonable and accurate. By following these steps, you should be able to produce accurate and well-structured outputs for the given inputs. Good luck!	0.69
DLN1-Momentum = 0.6	1. ‘question‘ (str) - Your output fields are: - 1. ‘reasoning‘ (str) - 2. ‘answer‘ (str) - All interactions will be structured in the following way, with the appropriate values filled in. [[## question ##]] question [[## reasoning ##]] reasoning [[## answer ##]] answer [[## completed ##]] - In adhering to this structure, your objective is: - Given the input ‘question‘, produce the fields ‘answer‘ by applying logical and mathematical operations to the provided information. - Ensure that your output is accurate and well-supported by your reasoning. - Note: - For calculations involving percentages, ratios, and proportions, use decimal equivalents. For example, 10% can be converted to 0.1. - For questions involving time, ensure accurate calculations of time periods and convert them to the required units. For example, if a time period is given in hours, convert it to days or minutes as needed. - Check your units and make sure they match the required units in the problem. - By following these guidelines, you will be able to produce accurate and well-supported answers. - Remember to: - Clearly explain your steps in the ‘reasoning‘ field. - Provide accurate calculations and assumptions. - Avoid ambiguity in your answers. - Ensure that your answer is concise and easy to understand. - For the ‘answer‘ field, provide the numerical value or solution to the problem. - For the ‘reasoning‘ field, provide a clear and concise explanation of your steps and calculations. - In the ‘completed‘ field, indicate that your task is finished and your output is ready for review. - By following these guidelines and the structure provided, you will be able to produce accurate and well-supported answers. - Note: - If you are unsure about any part of the problem or calculation, it is recommended that you review the problem and recalculate your answer before submitting it. - It is also important to double-check your units and ensure that they match the required units in the problem. - If you have any questions or concerns, please don’t hesitate to ask for help.	0.81

Table 34: Prompt comparison for the **GSM8K**(test) task, showing human, DLN1, and DLN1-Momentum prompts. Colored text highlights components such as Task Description, Term Clarification, Solution Guidance and Priority/Emphasis. Prompts generated by DLN1-Momentum are more detailed, structured, and pedagogically grounded, often mixing Solution Guidance and Priority/Emphasis to better support task completion and reduce ambiguity.

F.5 Iterative Workflow with Momentum based sampling

To illustrate the effectiveness of our momentum sampling module as an enhancement to the vanilla TextGrad framework, we use DLN1 as a representative method. We report all meta prompts generated by both DLN1 and DLN1-Momentum ($= 0.6$) throughout the iterative optimization process on the subjective classification task. Specifically, we present the evolving meta prompts—defined as the best-performing prompts selected from a candidate pool at each iteration. Only the changing meta prompts are shown under the H_1 setting, where the prompt generation temperature is set to 1.1 and early stopping is triggered if the best test accuracy does not improve over 5 consecutive iterations.

Compared to the standard DLN1 framework, DLN1-Momentum$= 0.6$ demonstrates a more gradual and stable improvement in test set accuracy across iterations. While DLN1 achieves a rapid performance boost in early iterations—jumping from 0.491 to 0.750 at iteration 1—its accuracy gains begin to plateau shortly thereafter. In contrast, DLN1-Momentum exhibits a slower but smoother trajectory, progressing from 0.491 to 0.66 in the first iteration and gradually increasing to 0.856 by iteration 5. This extended optimization process allows the momentum-based approach to explore a wider space of meta prompts and refine them more cautiously. As a result, DLN1-Momentum tends to converge toward stronger final prompts with higher accuracy, suggesting that the momentum mechanism effectively stabilizes prompt updates and reduces premature convergence on suboptimal prompt candidates.

Meta Prompt #	Meta Prompt	Test Acc.
0	Classify the input text as subjective or objective.	0.491
1	For each input text, classify whether the statement is a fact (objective) or an opinion/emotion (subjective). If the statement is describing a situation, event, or thing, it is likely an objective statement. If the statement is expressing a personal view, feeling, or evaluation, it is likely a subjective statement. Consider the language used, tone, and context to make your decision. Pay attention to words such as "is", "seems", "feels", "right", and "wrong."	0.750
2	1. Read the input text carefully. 2. Identify whether the text describes a situation, event, or thing (objective) or expresses a personal view, feeling, or evaluation (subjective). 3. Pay attention to words like "is," "seems," "feels," "right," and "wrong." 4. Consider the language used and the context in which it is used.	0.756
3	1. Read the input text carefully. 2. Identify whether the text describes a situation, event, or thing (objective) or expresses a personal view, feeling, or evaluation (subjective). Pay attention to words like "is," "seems," "feels," "right," and "wrong." 3. Consider the language used and the context in which it's used.	0.8125
4	1. Read the text carefully. 2. Determine whether the text is describing a situation, event, or thing (objective) or expressing a personal view, feeling, or evaluation (subjective). Pay attention to words like "is," "seems," "feels," "right," and "wrong." 3. Use context clues like language, tone, and the presence of emotions to help determine the text type.	0.825

Table 35: DLN1 Iterative Prompt Workflow for Subj Task

Meta Prompt #	Meta Prompt	Test Acc.
0	Classify the input text as subjective or objective.	0.491
1	1. Read the input text carefully. 2. Determine if the text is describing an opinion, a feeling, or an evaluation about something. If it is, classify it as subjective. If it is describing a fact, situation, or event without expressing a personal opinion, classify it as objective. 3. Check if the text includes words or phrases that indicate an evaluation, such as "good", "bad", "interesting", "amazing", or phrases that express a personal point of view	0.66
2	1. Read the input text carefully and identify the purpose of the text. 2. Ask yourself, "Is the text giving information about a fact, situation, or event without expressing a personal opinion?" If yes, classify it as objective. If not, consider whether the text is expressing an evaluation, opinion, or personal point of view. 3. Look for words or phrases that indicate an evaluation, such as "good", "bad", "interesting", "amazing", or phrases that	0.78
3	1. Read the text carefully and identify the purpose of the text. Does the text present a fact, situation, or event without expressing a personal opinion? Look for words or phrases that indicate an evaluation, such as "good", "bad", "interesting", or phrases that imply a subjective perspective. 2. Ask yourself, "Is the text describing a situation or event without making a value judgment?" If yes, classify it as objective. If not, consider whether the text is expressing an evaluation	0.8
4	1. Determine whether the text presents a fact, situation, or event without expressing a personal opinion. Look for words or phrases that indicate an objective description, such as "the film is a comedy" or "the main character is a woman". 2. Check for words or phrases that imply a subjective perspective, such as "good", "bad", "interesting", or "brilliant". 3. Ask yourself, "Does the text describe the event or situation without expressing an opinion?"	0.85
5	1. Determine whether the text presents a fact, situation, or event without expressing a personal opinion. Look for objective words or phrases that describe a description, such as "the film is a comedy" or "the main character is a woman". Avoid subjective words or phrases that convey a personal perspective, such as "good", "bad", "interesting", or "brilliant". If you find a subjective phrase, it likely expresses an opinion and is not a simple fact or description.	0.856

Table 36: DLN1-Momentum=0.6 Iterative Prompt Workflow for Subj Task