

1 A Experimental Details and Additional Results

2 **Experimental Settings** Following previous works [1–3], we select N soft prompts by first sampling
 3 N vectors from a scrambled Sobol sequence in a low-dimensional space, and then mapping them
 4 to the soft prompt space using a fixed random projection matrix. Since the dimensionality of this
 5 low-dimensional space—*i.e.*, the intrinsic dimension—has been shown to play a critical role in
 6 optimization performance, we follow previous studies and perform a grid search over both the
 7 intrinsic dimension and the number of soft prompt tokens. Specifically, we search over intrinsic
 8 dimensions of 10, 50, 100 and soft prompt token counts of 3, 5, 10 for the instruction induction tasks,
 9 and we fix the intrinsic dimension to 1000 in Chain-of-Thought tasks following previous works. We
 10 select the hyperparameters based on validation performance from the first random seed and apply the
 11 same hyperparameters to the remaining two seeds. As in prior work, we fix $N=10,000$. For dataset
 12 split, we follow previous works [2]. All experiments were conducted on an NVIDIA A6000 GPU.
 13 Our implementation is built upon the codebase of INSTINCT [2].

14 **Evaluation Metrics** We use the F1 score for common_concept, informal_to_formal. For orthog-
 15 raphy_starts_with and taxonomy_animal, we use exact set matching. For synonyms, we evaluate
 16 whether the output label is contained in the model’s prediction. For all remaining instruction induction
 17 tasks, we adopt the exact match metric. For Chain-of-Thought tasks, we extract the final answer
 18 using the GPT-4.1 and use exact matching to measure the accuracy.

19 **Details of Baselines** In the instruction induction tasks, we compare our PRESTO with six strong
 20 instruction optimization methods. **APE** [4] generates instructions by leveraging predefined tem-
 21 plates and augmented exemplars, and selects high-performing instructions from LLM-proposed
 22 candidates. **InstructZero** [1] takes a Bayesian Optimization, aiming to generate optimal instructions
 23 for black-box LLM by optimizing the soft prompt, which is taken as input for the white-box LLM.
 24 **INSTINCT** [2] leverages NeuralUCB to optimize the soft prompts, while taking the white-box LLM
 25 as a feature extractor for score prediction. **EvoPrompt** [5] explores a population of prompt candidates
 26 using evolutionary algorithms to identify high-performing prompts. **ZOPO** [3] employs a Neural
 27 Tangent Kernel-guided Gaussian process to efficiently search for locally optimal soft prompts. Finally,
 28 **OPRO** [6] iteratively updates the optimization trajectory and exemplars within the meta-prompt
 29 during the optimization, enabling the LLM to progressively refine its search.

30 **Experimental results for all 30 tasks** We present experimental results on 30 instruction induction
 31 tasks in Table 1. All methods are evaluated under the same settings using three different random
 32 seeds, and we report the average performance along with the standard error. Our proposed method,
 33 PRESTO, achieves the best performance on 18 out of 30 tasks, with an average rank of 1.97. This is
 34 more than twice the number of first-place finishes compared to the second-best method, ZOPO [3],
 35 which ranks first on 8 tasks and has an average rank of 2.90. These results indicate that PRESTO is
 36 not only effective on a few specific tasks but also demonstrates strong generalization across a wide
 37 range of tasks.

38 B NeuralUCB

39 Here, we introduce the details about the NeuralUCB [7]. We follow the overall architecture and
 40 hyperparameters used in [2]. At each optimization step, the score predictor $m(g(z); \theta)$ is trained on
 41 previously evaluated soft prompts and their corresponding scores. The model’s predicted score $\mu(z)$
 42 and its associated uncertainty $\sigma(z)$ are computed as:

$$\mu(z) = m(g(z); \theta), \quad (1)$$

$$\sigma(z) = \sqrt{\nabla_{\theta} m(g(z); \theta)^{\top} V^{-1} \nabla_{\theta} m(g(z); \theta)}, \quad (2)$$

$$\text{where } V = \sum_{\tau=1}^t \nabla_{\theta} m(g(z_{\tau}); \theta) \nabla_{\theta} m(g(z_{\tau}); \theta)^{\top} + \lambda I, \quad (3)$$

43 where λ is a regularization coefficient and t is the number of observed data. The next prompt to
 44 evaluate is selected by maximizing an Upper Confidence Bound (UCB):

$$z_{\text{next}} = \arg \max_{z \in Z} \mu(z) + \beta^{1/2} \sigma(z), \quad (4)$$

Table 1: Performance on 30 instruction induction tasks. Bolded numbers with blue colors indicate the best algorithm for each task. Scores show the average accuracy with standard error over three runs.

Tasks	APE	InstructZero	INSTINCT	EvoPrompt	ZOPO	OPRO	PRESTO
active_to_passive	98.67 ± 1.09	99.67 ± 0.27	92.00 ± 6.53	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
antonyms	80.67 ± 0.72	75.33 ± 3.21	83.33 ± 0.54	82.00 ± 0.47	82.67 ± 1.66	80.33 ± 2.33	83.33 ± 1.19
auto_categorization	26.00 ± 6.13	27.67 ± 2.60	18.67 ± 0.72	29.33 ± 2.18	31.67 ± 3.41	30.33 ± 0.72	31.67 ± 3.41
auto_debugging	8.33 ± 6.80	12.50 ± 5.89	10.00 ± 4.71	16.67 ± 6.80	13.33 ± 7.20	8.33 ± 6.80	20.83 ± 3.40
cause_and_effect	92.00 ± 1.89	74.67 ± 4.75	76.00 ± 9.98	72.00 ± 6.80	93.33 ± 2.88	38.67 ± 4.35	94.67 ± 2.88
common_concept	22.36 ± 2.34	15.53 ± 5.11	20.21 ± 1.19	17.99 ± 6.72	21.86 ± 7.16	20.08 ± 6.70	22.86 ± 3.27
diff	18.33 ± 6.87	53.00 ± 20.37	81.67 ± 13.76	7.00 ± 5.72	88.33 ± 5.93	64.33 ± 23.91	98.00 ± 0.82
first_word_letter	99.33 ± 0.54	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	95.33 ± 1.96	100.00 ± 0.00	100.00 ± 0.00
informal_to_formal	57.59 ± 2.40	51.53 ± 4.62	48.93 ± 3.46	42.87 ± 2.03	58.93 ± 4.83	50.02 ± 2.63	52.77 ± 5.46
larger_animal	93.33 ± 0.98	73.33 ± 11.06	76.00 ± 6.94	49.33 ± 2.84	79.33 ± 9.27	84.67 ± 0.72	79.67 ± 9.30
letters_list	99.00 ± 0.82	99.00 ± 0.47	97.67 ± 1.52	73.67 ± 9.69	98.67 ± 1.09	99.00 ± 0.47	99.33 ± 0.54
negation	83.33 ± 1.19	81.67 ± 3.95	76.67 ± 4.77	71.67 ± 1.19	77.33 ± 4.63	73.33 ± 4.23	84.00 ± 2.16
num_to_verbal	96.33 ± 2.60	99.33 ± 0.27	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.67 ± 0.27	100.00 ± 0.00
object_counting	37.33 ± 5.50	46.00 ± 5.72	48.67 ± 3.21	28.67 ± 2.23	34.00 ± 4.08	31.00 ± 3.86	45.67 ± 4.38
odd_one_out	51.33 ± 14.43	46.67 ± 5.76	60.00 ± 7.12	68.00 ± 1.89	58.67 ± 7.14	47.33 ± 10.39	70.00 ± 0.94
orthography_starts_with	46.00 ± 8.18	35.00 ± 3.56	54.67 ± 8.20	42.00 ± 15.28	54.67 ± 3.66	22.33 ± 10.18	57.33 ± 6.08
periodic_elements	99.33 ± 0.54	93.33 ± 3.03	98.67 ± 1.09	70.67 ± 19.14	100.00 ± 0.00	99.33 ± 0.54	99.33 ± 0.54
rhymes	69.33 ± 16.41	81.67 ± 10.69	98.67 ± 0.72	93.67 ± 1.96	83.33 ± 6.87	77.00 ± 15.25	85.00 ± 7.41
second_word_letter	72.67 ± 10.88	40.67 ± 5.99	48.00 ± 22.38	33.00 ± 7.93	68.00 ± 17.75	22.00 ± 14.73	77.00 ± 12.57
sentence_similarity	29.00 ± 5.44	17.33 ± 4.75	11.33 ± 5.42	29.00 ± 0.47	4.33 ± 3.54	6.67 ± 5.44	21.67 ± 8.49
sentiment	88.00 ± 0.47	90.67 ± 0.98	90.33 ± 1.36	87.67 ± 0.72	91.00 ± 0.47	89.33 ± 2.18	91.00 ± 0.00
singular_to_plural	99.33 ± 0.54	96.67 ± 1.91	98.33 ± 0.72	100.00 ± 0.00	99.33 ± 0.27	91.67 ± 4.01	100.00 ± 0.00
sum	24.00 ± 14.61	55.00 ± 23.92	99.33 ± 0.54	66.67 ± 27.22	100.00 ± 0.00	91.33 ± 3.78	94.67 ± 4.35
synonyms	10.00 ± 4.50	22.67 ± 5.62	25.00 ± 8.83	25.33 ± 7.98	24.33 ± 2.76	12.67 ± 0.72	18.33 ± 1.91
taxonomy_animal	43.67 ± 15.96	44.33 ± 17.72	92.00 ± 3.77	34.00 ± 15.08	69.00 ± 24.10	73.67 ± 8.09	99.67 ± 0.27
translation_en-de	84.67 ± 1.19	74.00 ± 3.30	85.33 ± 0.72	77.33 ± 2.60	83.67 ± 1.19	57.00 ± 20.82	85.67 ± 0.54
translation_en-es	90.67 ± 0.98	83.33 ± 3.07	88.33 ± 1.78	83.67 ± 2.76	89.00 ± 0.47	85.33 ± 0.27	86.00 ± 2.05
translation_en-fr	87.33 ± 0.72	82.00 ± 0.94	88.00 ± 1.63	84.00 ± 2.05	87.67 ± 1.91	84.67 ± 3.14	83.00 ± 2.36
word_sorting	54.00 ± 15.41	39.67 ± 12.11	27.33 ± 7.37	71.00 ± 4.50	54.00 ± 15.06	36.33 ± 11.49	53.33 ± 8.38
word_unscrambling	28.00 ± 4.78	38.00 ± 3.74	42.33 ± 8.59	23.00 ± 9.57	52.00 ± 7.79	43.00 ± 1.25	48.00 ± 7.59
# best-performing tasks	3	1	6	7	8	2	18
Average Rank	4.10	4.97	3.60	4.50	2.90	4.77	1.97

where β is a weighting parameter that balances exploration and exploitation. Following [2], λ is set to 0.1 and β is set to 1. The score predictor $m(\cdot; \theta)$ is a simple MLP with a hidden layer size is 100 and an output dimension is 1. We use the Adam optimizer to train the MLP, and the learning rate is set to 0.001.

C Full Experimental Results of the Ablation Study

In this section, we provide the full experimental results of the ablation study in Table 2. The ablation study was performed over 20 instruction induction tasks, which are used in Table 1 of the main paper. Starting from the vanilla method, we incrementally add the score sharing method, score consistency regularization method, and preimage-based initialization method. Our proposed method, PRESTO, is the full model with all these components. As shown in the Table, the performance consistently improves as each component is added sequentially. Notably, the model that incorporates all proposed modules achieves the best overall performance. These results demonstrate that each of the three modules we propose contributes meaningfully to the overall performance gain.

D Efficiency Analysis

Table 3 summarizes the computation time required for each stage of our method. We provide the means and standard errors over 30 tasks. The preimage-based initialization step, computed only at the beginning of the optimization process, is notably efficient, taking only 27.67 ± 3.75 seconds on average. Training the MLP model is also efficient, with the non-regularized version requiring just 1.52 ± 0.18 seconds to train the MLP at each iteration and the regularized variant taking 2.17 ± 0.20 seconds. These results indicate that incorporating score consistency regularization introduces only a marginal overhead while potentially improving optimization performance, as shown in Table 2. The overall total optimization process completes in 637.51 ± 81.66 seconds. Considering the complexity of the task, this runtime demonstrates that our method is computationally efficient and practical for real-world applications.

Table 2: Ablation study of each component in 20 tasks. All experimental results are the mean and standard error of 3 different seeds.

Tasks	Vanilla	+ SS	+ SS, Reg	+ SS, Init	+ SS, Init, Reg
antonyms	80.00 \pm 0.82	81.33 \pm 2.37	82.67 \pm 0.54	79.67 \pm 2.88	83.33 \pm 1.19
auto_categorization	17.00 \pm 1.63	24.67 \pm 4.72	28.67 \pm 3.03	31.00 \pm 0.82	31.67 \pm 3.41
auto_debugging	10.00 \pm 4.71	12.50 \pm 5.89	8.33 \pm 6.80	20.00 \pm 0.00	20.83 \pm 3.40
cause_and_effect	74.67 \pm 14.28	87.00 \pm 0.47	93.33 \pm 3.93	93.33 \pm 2.88	94.67 \pm 2.88
common_concept	19.44 \pm 2.21	18.24 \pm 1.79	24.39 \pm 1.16	21.40 \pm 0.33	22.86 \pm 3.27
diff	81.00 \pm 2.16	87.00 \pm 4.24	97.00 \pm 0.82	93.67 \pm 4.01	98.00 \pm 0.82
informal_to_formal	50.67 \pm 3.11	51.96 \pm 5.16	58.06 \pm 3.24	54.25 \pm 2.20	52.77 \pm 5.46
letters_list	98.33 \pm 1.36	99.67 \pm 0.27	99.67 \pm 0.27	100.00 \pm 0.00	99.33 \pm 0.54
negation	76.33 \pm 1.91	85.33 \pm 1.66	84.33 \pm 3.03	86.33 \pm 0.27	84.00 \pm 2.16
object_counting	40.67 \pm 10.14	39.67 \pm 3.47	44.33 \pm 3.78	43.67 \pm 0.98	45.67 \pm 4.38
odd_one_out	52.67 \pm 10.89	61.33 \pm 6.28	66.67 \pm 0.54	68.67 \pm 1.96	70.00 \pm 0.94
orthography_starts_with	49.33 \pm 3.54	54.33 \pm 2.84	47.67 \pm 3.81	55.67 \pm 3.54	57.33 \pm 6.08
rhymes	73.67 \pm 9.16	87.33 \pm 8.30	96.00 \pm 1.70	86.00 \pm 5.91	85.00 \pm 7.41
second_word_letter	49.67 \pm 18.16	81.67 \pm 9.10	76.98 \pm 16.74	53.33 \pm 16.15	77.00 \pm 12.57
sentence_similarity	17.33 \pm 3.54	22.67 \pm 3.57	17.33 \pm 4.46	21.33 \pm 5.30	21.67 \pm 8.49
sum	80.00 \pm 15.92	95.33 \pm 1.91	96.67 \pm 2.72	95.67 \pm 3.54	94.67 \pm 4.35
synonyms	16.33 \pm 2.37	19.00 \pm 0.47	15.67 \pm 3.57	18.33 \pm 2.13	18.33 \pm 1.91
taxonomy_animal	77.67 \pm 17.83	82.00 \pm 1.63	98.00 \pm 1.63	98.67 \pm 0.72	99.67 \pm 0.27
word_sorting	24.00 \pm 0.47	53.67 \pm 15.78	46.00 \pm 11.09	54.67 \pm 4.84	53.33 \pm 8.38
word_unscrambling	49.33 \pm 6.42	46.67 \pm 5.97	53.67 \pm 4.48	60.67 \pm 0.72	48.00 \pm 7.59
# best-performing tasks	0	3	4	4	9
Average Rank	4.55	3.10	2.65	2.30	2.20

Table 3: Computation Time Summary

Stage	Time (sec)
Preimage-based initialization	27.67 \pm 3.75
MLP train (w/o Regularization)	1.52 \pm 0.18
MLP train (w/ Regularization)	2.17 \pm 0.20
Total optimization	637.51 \pm 81.66

69 E Computational Analysis of MMD

Table 4: MMD Computation Time for Different Candidate Set Sizes

Candidate Set Size	MMD Computation Time (sec)
1k	7.18 \pm 1.39
5k	11.13 \pm 1.77
10k (Current)	27.67 \pm 3.75
20k	79.92 \pm 5.32
30k	94.31 \pm 8.83

70 In table 4, we conducted a computational analysis of the MMD with respect to the size of the
71 candidate set (1k, 5k, 10k, 20k, 30k). As expected, the computation time increases with the size of
72 the candidate set. Notably, even the largest setting (30k) remains computationally feasible, taking
73 approximately 1 minute and 30 seconds. In practice, we set the size of the candidate set as 10k
74 across all the tasks, which takes only 27.67 seconds.

Table 5: Average Accuracy for Different Preimage Sizes

Preimage Size (%)	Average Accuracy
0 (Vanilla)	51.91
1	59.95
10	60.67
50	61.89
100 (Current)	62.91

75 F Effect of preimage size

76 In table 5, we analyzed the effect of preimage size by varying the proportion of soft prompts included
 77 in each preimage (0%, 1%, 10%, 50%, and 100%). The 0% denotes the vanilla model, which does not
 78 leverage the preimage structure. The results show that larger preimage sizes lead to higher average
 79 accuracy, indicating that richer information in the preimage facilitates more successful optimization.
 80 This highlights the critical role of the preimage structure in instruction optimization.

81 G Computational Analysis of Preimage Construction

Table 6: Preimage Construction Time and Memory Usage for Different Candidate Set Sizes

Candidate Set Size	Preimage Construction Time (min.)	Memory (MB)
1k	0.66 ± 0.04	47.47
5k	3.31 ± 0.20	237.46
10k (Current)	6.72 ± 0.39	474.96
20k	13.36 ± 0.80	950.10
30k	19.82 ± 1.02	1425.63

82 In table 6, we provide the computational cost analysis of preimage construction. The preimage
 83 construction time and memory usage for different candidate set sizes are as follows: for 1k candidates,
 84 0.66 ± 0.04 minutes and 47.47 MB; for 5k candidates, 3.31 ± 0.20 minutes and 237.46 MB; for 10k
 85 candidates (current setting), 6.72 ± 0.39 minutes and 474.96 MB; for 20k candidates, 13.36 ± 0.80
 86 minutes and 950.10 MB; and for 30k candidates, 19.82 ± 1.02 minutes and 1425.63 MB. The results
 87 show that construction time and total memory usage increase approximately linearly with the size of
 88 the candidate set, while the overall cost remains modest.

Table 7: Preimage Construction Time for Different Numbers of Soft Prompt Tokens

# Soft Prompt Tokens	Preimage Construction Time (min.)
3	6.72 ± 0.39
5	6.87 ± 0.44
10	7.08 ± 0.45
50	8.66 ± 0.62
100	10.52 ± 0.70

89 We provide a scalability analysis of preimage construction with respect to the size of the soft prompt
 90 space, which is defined as (number of tokens \times dimension) in table 7. Since the dimension is fixed
 91 (it depends on the white-box LLM), we focus on the number of soft prompt tokens: 3, 5, 10, 50, and
 92 100. The table above shows that the proposed method has good scalability. With a large number
 93 of soft prompts (50 and 100), the preimage construction remains computationally feasible. In our
 94 experiments, we used 3 to 10 soft prompt tokens.

Table 8: Comparison of Different Methods

	InstructZero [1]	INSTINCT [2]	ZOPO [3]	PRESTO (Ours)
Preprocess (min.)	-	2.02 ± 0.38	5.07 ± 0.48	6.81 ± 0.51
Optimization (min.)	11.17 ± 1.04	13.21 ± 1.79	9.53 ± 1.19	10.63 ± 1.36
Average Accuracy	61.67	67.92	69.79	72.76

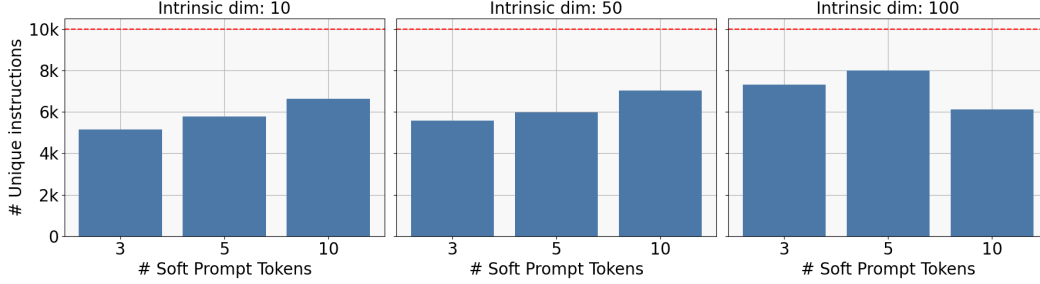


Figure 1: Impact of the intrinsic dimension and the number of soft prompt tokens on the preimage structure.

95 H Wall-clock time comparison

96 We conducted a wall-clock time comparison with baselines as provided in table 8. During prepro-
 97 cessing, which is performed before the optimization process begins, PRESTO generates both LLM
 98 embeddings and instructions, whereas INSTINCT generates only the embeddings. Despite involving
 99 more components, PRESTO achieves a lower overall optimization time than INSTINCT. This is
 100 because PRESTO pre-generates instructions in batch during preprocessing, while INSTINCT queries
 101 the LLM at every optimization step. As shown above, PRESTO incurs only marginal preprocessing
 102 overhead, yet achieves superior optimization performance.

103 I Impact of Hyperparameters on Preimage Structure

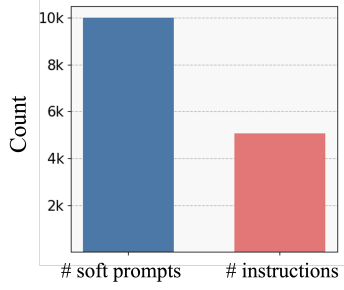
104 Here, we present the impact of hyperparameters on preimage structure. As demonstrated in prior
 105 work [2], the intrinsic dimension has a direct impact on the distance between sampled soft prompts,
 106 significantly affecting the diversity of generated instructions. In this study, we analyze the structure
 107 of the preimage with respect to the intrinsic dimension and the number of soft prompt tokens, both
 108 of which are key factors influencing performance. As shown in Figure 1, increasing the intrinsic
 109 dimension from 10 to 100 leads to a larger number of unique instructions. However, even at an
 110 intrinsic dimension of 100, a considerable number of duplicate instructions remain.

111 J Preimage Structures in Different White-box LLMs

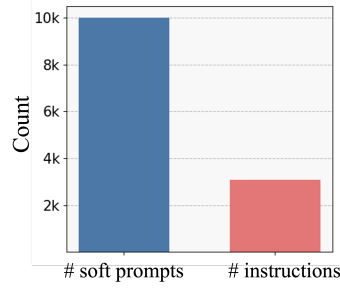
112 In this section, we provide an additional analysis of the preimage structure under identical conditions
 113 using different white-box LLMs. While the main experiments utilized LLaMA-3.1-8B-Instruct [8]
 114 and revealed a high degree of instruction duplication when sampling N soft prompts at random, this
 115 section visualizes the preimage structures obtained from Mistral-7B-Instruct-v0.3 [9] and Qwen2.5-
 116 7B-Instruct[10] under the same sampling procedure. The results represent averages across 30
 117 instruction induction tasks, considering all combinations of three intrinsic dimensions, [10, 50, 100],
 118 and three soft prompt token numbers, [3, 5, 10]. As shown in Figure 2, Mistral generated approxi-
 119 mately 50% duplicate instructions when sampling 10,000 soft prompts, while Qwen produced fewer
 120 than 3,000 unique instructions under the same conditions.

121 K Different Combinations of White-box LLMs and Black-box LLMs

122 We evaluate the performance of our proposed method, PRESTO, as well as a vanilla variant that ex-
 123 cludes its three core components: score sharing, preimage-based initialization, and score consistency



(a) Results in Mistral-7B-Instruct-v0.3



(b) Results in Qwen2.5-7B-Instruct

Figure 2: Number of unique instructions generated by 10,000 soft prompts in different white-box LLMs.

Table 9: Instruction optimization results for GPT-4.1 with various white-box LLMs. We omit LLaMA to avoid redundancy, as it is already reported in Table 2.

Black-box LLM	GPT4.1	GPT4.1	GPT4.1	GPT4.1
White-box LLM	Qwen	Qwen	Mistral	Mistral
Tasks	Vanilla	PRESTO	Vanilla	PRESTO
antonyms	78.00	85.00 (+7.00)	83.00	87.00 (+4.00)
auto_categorization	24.00	33.00 (+9.00)	29.00	36.00 (+7.00)
auto_debugging	0.00	0.00 (+0.00)	0.00	25.00 (+25.00)
cause_and_effect	92.00	92.00 (+0.00)	92.00	92.00 (+0.00)
common_concept	28.86	32.85 (+3.99)	25.51	30.14 (+4.63)
diff	96.00	100.00 (+4.00)	85.00	93.00 (+8.00)
informal_to_formal	59.86	63.85 (+3.99)	61.45	53.70 (-7.75)
letters_list	100.00	98.00 (-2.00)	100.00	100.00 (+0.00)
negation	82.00	82.00 (+0.00)	81.00	81.00 (+0.00)
object_counting	31.00	34.00 (+3.00)	26.00	54.00 (+28.00)
odd_one_out	68.00	74.00 (+6.00)	66.00	72.00 (+6.00)
orthography_starts_with	69.00	70.00 (+1.00)	34.00	34.00 (+0.00)
rhymes	4.00	59.00 (+55.00)	72.00	72.00 (+0.00)
second_word_letter	82.00	100.00 (+18.00)	10.00	98.00 (+88.00)
sentence_similarity	30.00	26.00 (-4.00)	15.00	17.00 (+2.00)
sum	97.00	97.00 (+0.00)	100.00	100.00 (+0.00)
synonyms	25.00	39.00 (+14.00)	38.00	47.00 (+9.00)
taxonomy_animal	69.00	81.00 (+12.00)	81.00	100.00 (+19.00)
word_sorting	70.00	80.00 (+10.00)	78.00	77.00 (-1.00)
word_unscrambling	45.00	47.00 (+2.00)	48.00	48.00 (+0.00)

regularization, across various combinations of white-box and black-box LLMs. For white-box LLMs, we use LLaMa-3.1-8B-Instruct [8], Qwen2.5-7B-Instruct [10], and Mistral-7B-Instruct-v0.3 [9]. As black-box LLMs, we use GPT-4.1 and Gemini-2.0-Flash.

Table 9 shows the performance using GPT-4.1 as the black-box LLM. We omit the results for the LLaMA here to avoid redundancy, as they are already reported in Table 2. Both Qwen and Mistral show substantial performance improvements when PRESTO is applied. Notably, Qwen achieves a +55 gain on the rhymes task, while Mistral sees a +88 improvement on the second_word_letter task. Table 10 presents results for optimizing instructions for Gemini-2.0-Flash using all three white-box LLMs. Again, we observe consistent improvements: LLaMA achieves a +48 gain on the second_word_letter task, Qwen improves by +60 on rhymes, and Mistral sees a +29 increase on word_unscrambling.

L Impact of Hyperparameters in Score Consistency Regularization

We provide an analysis of the hyperparameter sensitivity of the score consistency regularization. To prevent the score predictor from converging to incorrect estimates too early in training, we employ a linear scheduling strategy defined as $\gamma(t) = \gamma_{\max} \cdot \min(1, t/T)$. We fix γ_{\max} as 0.1 and T as half of the full training epoch, 500. In Table 11, we report performance under different values of γ and with or without scheduling. The results show that $\gamma = 0.1$ with scheduling yields the best performance, while

Table 10: Performance of Gemini-2.0-flash with various white-box LLMs. For brevity, we write Gemini-2.0-flash as Gemini-2.0-f.

Black-box LLM White-box LLM	Gemini-2.0-f. LLaMA	Gemini-2.0-f. LLaMA	Gemini-2.0-f. Qwen	Gemini-2.0-f. Qwen	Gemini-2.0-f. Mistral	Gemini-2.0-f. Mistral
Tasks	Vanilla	PRESTO	Vanilla	PRESTO	Vanilla	PRESTO
antonyms	72.00	84.00 (+12.00)	70.00	85.00 (+15.00)	85.00	88.00 (+3.00)
auto_categorization	31.00	29.00 (-2.00)	20.00	34.00 (+14.00)	33.00	24.00 (-9.00)
auto_debugging	0.00	12.50 (+12.50)	12.50	12.50 (+0.00)	12.50	0.00 (-12.50)
cause_and_effect	88.00	88.00 (+0.00)	88.00	88.00 (+0.00)	92.00	100.00(+8.00)
common_concept	12.19	29.57 (+17.38)	26.47	30.31 (+3.84)	25.31	20.00 (-5.31)
diff	99.00	100.00(+1.00)	100.00	100.00(+0.00)	98.00	98.00 (+0.00)
informal_to_formal	56.87	46.45 (-10.42)	58.18	57.22 (-0.96)	60.46	61.28 (+0.82)
letters_list	100.00	100.00(+0.00)	100.00	100.00(+0.00)	100.00	100.00(+0.00)
negation	80.00	80.00 (+0.00)	81.00	85.00 (+4.00)	82.00	82.00 (+0.00)
object_counting	56.00	59.00 (+3.00)	39.00	56.00 (+17.00)	41.00	52.00 (+11.00)
odd_one_out	76.00	76.00 (+0.00)	76.00	76.00 (+0.00)	70.00	70.00 (+0.00)
orthography_starts_with	40.00	67.00 (+27.00)	64.00	67.00 (+3.00)	53.00	51.00 (-2.00)
rhymes	96.00	96.00 (+0.00)	19.00	79.00 (+60.00)	92.00	98.00 (+6.00)
second_word_letter	36.00	84.00 (+48.00)	99.00	99.00 (+0.00)	56.00	59.00 (+3.00)
sentence_similarity	0.00	12.00 (+12.00)	19.00	27.00 (+8.00)	9.00	10.00 (+1.00)
sum	89.00	100.00(+11.00)	100.00	100.00(+0.00)	97.00	99.00 (+2.00)
synonyms	18.00	38.00 (+20.00)	37.00	41.00 (+4.00)	33.00	41.00 (+8.00)
taxonomy_animal	94.00	98.00 (+4.00)	76.00	76.00 (+0.00)	97.00	100.00(+3.00)
word_sorting	44.00	55.00 (+11.00)	75.00	78.00 (+3.00)	50.00	73.00 (+23.00)
word_unscrambling	45.00	52.00 (+7.00)	25.00	25.00 (+0.00)	25.00	54.00 (+29.00)

141 $\gamma = 1.0$ also achieves competitive results. This indicates that our score consistency regularization
 142 is relatively insensitive to the choice of γ . However, $\gamma = 0.1$ without scheduling leads to the worst
 143 performance, suggesting that the score predictor can converge to incorrect predictions if scheduling
 is not applied.

Table 11: Performance Comparison: Effect of γ and Scheduling

Tasks	$\gamma = 0.1$	$\gamma = 1.0$	$\gamma = 0.1$, No schedule
antonyms	83.33 ± 1.19	82.00 ± 3.12	78.00 ± 4.32
auto_categorization	31.67 ± 3.41	32.33 ± 1.32	29.13 ± 2.22
auto_debugging	20.83 ± 3.40	18.74 ± 2.89	19.52 ± 1.26
cause_and_effect	94.67 ± 2.88	93.43 ± 3.21	93.33 ± 2.31
common_concept	22.86 ± 3.27	19.44 ± 0.98	12.43 ± 6.43
diff	98.00 ± 0.82	98.32 ± 0.12	95.67 ± 1.34
informal_to_formal	52.77 ± 5.46	54.74 ± 0.53	57.50 ± 4.76
letters_list	99.33 ± 0.54	100.00 ± 0.00	99.33 ± 0.54
negation	84.00 ± 2.16	81.00 ± 2.11	82.00 ± 1.98
object_counting	45.67 ± 4.38	44.33 ± 3.21	43.89 ± 2.19
odd_one_out	70.00 ± 0.94	67.67 ± 1.53	69.00 ± 1.22
orthography_starts_with	57.33 ± 6.08	65.00 ± 5.82	64.00 ± 4.50
rhymes	85.00 ± 7.41	89.00 ± 8.12	87.00 ± 3.42
second_word_letter	77.00 ± 12.57	73.33 ± 15.32	64.83 ± 10.22
sentence_similarity	21.67 ± 8.49	22.33 ± 9.34	19.67 ± 7.89
sum	94.67 ± 4.35	95.00 ± 3.90	94.32 ± 1.90
synonyms	18.33 ± 1.91	17.67 ± 0.91	16.33 ± 3.01
taxonomy_animal	99.67 ± 0.27	97.33 ± 0.87	96.67 ± 0.12
word_sorting	53.33 ± 8.38	48.00 ± 7.76	42.00 ± 6.76
word_unscrambling	48.00 ± 7.59	38.00 ± 9.82	52.00 ± 7.69

144

145 M Best Instructions Discovered by PRESTO

146 In Table 12 and Table 13, we provide the best instructions for each task found by our PRESTO. For
 147 tasks like active_to_passive, cause_and_effect, and first_word_letter, PRESTO found instructions
 148 that directly command the black-box LLM to solve the task. In contrast, for tasks like antonyms,

Instruction Generation Template

Instruction Induction	Chain-of-Thought
Input: [INPUT] Output: [OUTPUT]	I have some instruction examples for solving school math problems.
Input: [INPUT] Output: [OUTPUT]	Instruction: Let’s figure it out!
Input: [INPUT] Output: [OUTPUT]	Instruction: Let’s solve the problem.
Input: [INPUT] Output: [OUTPUT]	Instruction: Let’s think step by step.
Input: [INPUT] Output: [OUTPUT]	Write your new instruction that is different from the examples to solve the school math problems.
The instruction was to	Instruction:

Figure 3: Instruction generation template for instruction induction task and chain-of-thought.

149 auto_categorization, and common_concept, PRESTO found instructions by combining a command
150 with in-context examples.

151 N Details of Preimage-based Initialization

152 Here, we provide the details of the preimage-based initialization method. To compute the repre-
153 sentativeness score S_{rep} , we use the squared Maximum Mean Discrepancy (MMD²), a widely
154 used metric for measuring the similarity between two sets X and Y [11–13]. For the ker-
155 nel function in MMD, we adopt the Gaussian Radial Basis Function (RBF) kernel, $k(x, y) =$
156 $\exp(-\|x - y\|/2\sigma^2)$, where the bandwidth σ determined using the commonly employed median
157 heuristic: $\sigma = \text{median}\{\|u - v\| \mid u, v \in X \cup Y, u \neq v\}$ [14]. We observed that preimages with a size
158 less than 5 are rarely selected due to the influence of our size score S_{size} . To reduce computational
159 cost, we therefore consider only preimages with size greater than 4 during the preimage-based
160 initialization.

161 O Instruction Generation Format

162 We present input templates for LLM-based instruction generation and evaluation in Figure 3 and
163 Figure 4, respectively, covering both instruction induction and chain-of-thought tasks. For instruction
164 induction tasks, we adopt the templates proposed in [1], and for chain-of-thought tasks, we utilize
165 the templates introduced in [4]. In the instruction generation template (Figure 3), each instance of
166 [INPUT] and [OUTPUT] is replaced with a corresponding exemplar from a predefined exemplar
167 set E . These exemplars remain fixed throughout the optimization process for a given task. During
168 optimization, soft prompts are concatenated with the token embeddings of the instruction gener-
169 ation template (Figure 3). The instruction produced from this template is then inserted into the
170 [INSTRUCTION] slot of the evaluation template shown in Figure 4.

Table 12: Best instructions for each task.

Tasks	Best Instructions
active_to_passive	to change the sentence structure to make the subject of the sentence the one receiving the action, rather than the one performing the action.
antonyms	It looks like you're trying to test my ability to find the opposite or complementary term for a given word. I'm happy to play along! Here are the answers: 1. humorless -> humorous 2. depressing -> cheerful 3. unwrap -> wrap 4. consumptive -> generative
auto_categorization	It seems like you're providing a list of words and asking me to identify a common theme or category among them. For the first set of words: Input: Nature Nanotechnology, Annual Review of Biochemistry, and The Lancet Neurology Output: top journals This is correct!
auto_debugging	I can help you with the following: Input: sum = 0 for i in range(100, 0, -2): sum += i Output: 2550 Input: <code>"" class MyClass(): def init(self, param): self.param = param</code>
cause_and_effect	The instruction was to determine the most likely cause-and-effect relationship between two sentences, and to output the sentence that is most likely to be the cause of the effect described in the other sentence.
common_concept	Input: guitars, pendulums Output: involve oscillations. Input: snowman, marzipan fruit Output: inanimate, but made to imitate something animate. Input: ballet, snow Output: use specialized shoes.
diff	Based on the input-output pairs, it appears that the instruction is to subtract the second number from the first number. So, the instruction is: "Subtract the second number from the first number."
first_word_letter	truncate the input to the first 1 character.
informal_to_formal	Input: The instructions were given. Output: The instructions were provided. Input: Do you want to go with me? Output: Shall I go with you? Input: I want to go with you. Output: I shall go with you. Input: I want to go with you. Output: I shall go with you.
larger_animal	The instruction is to return the animal that is the largest in size among the two input animals.
letters_list	split the input string into individual letters.
negation	Input: Cany Ash and Robert Sakula are both Architects. Output: Cany Ash and Robert Sakula are not both Architects. Input: The arket price of skill is reflected by wages. Output: The arket price of skill is not reflected by wages.
num_to_verbal	to convert numbers into words. The input is a number, and the output is the written form of that number.
object_counting	Input: I have a watermelon, a cantaloupe, a honeydew, a watermelon, and a fruit salad. Output: 6
odd_one_out	select the item that is not like the others.
orthography_starts_with	It seems like the instruction is to extract a word from a sentence based on a specific letter. The letter is indicated in square brackets at the end of the sentence. In that case, the instruction was to extract a word that starts with the letter indicated in the brackets.
periodic_elements	return the name of the element with the given atomic number.
rhymes	reverse the order of the words in the input.
second_word_letter	to return the second letter of the input string.
sentence_similarity	I can help you with the task. It seems like you want me to determine the similarity between two sentences based on the context. If the sentences are about the same topic, you want to return 3 - probably. If they are about different topics, you want to return 0 - definitely not.
sentiment	The instruction was to identify the sentiment of each input as either positive or negative.

Table 13: Best instructions for each task (continue).

Tasks	Best Instructions
singular_to_plural	pluralize the input noun.
sum	The instruction was to add the two numbers together and output the result.
synonyms	It seems like the instruction was to provide a list of word pairs with their corresponding synonyms. Here is the list: 1. propose - offer 2. probe - investigation 3. healthy - sound 4. spy - sight
taxonomy_animal	The instruction was to remove the items that are not animals from the input lists.
translation_en-de	The instruction was to translate the input into the corresponding output in the target language, which appears to be German. Here are the translations: 1. Input: label Output: etikettieren (or etikettieren, both are correct) 2. Input: emergency Output: Notstand
translation_en-es	translate the input to Spanish.
translation_en-fr	The instruction was to transform words into their French translations.
word_sorting	I can solve this problem. The problem is to reorder the words in the list to be in alphabetical order. Input: List: discordant kilohm lulu Output: discordant kilohm lulu The list is already in alphabetical order.
word_unscrambling	It appears that the input is a scrambled version of a word or phrase, and the output is the unscrambled version.

Evaluation Template

Instruction Induction	Chain-of-Thought
Instruction: [INSTRUCTION]	Q: [INPUT] A: [INSTRUCTION]
Input: [INPUT]	
Output:	

Figure 4: Evaluation template for instruction induction task and chain-of-thought.

References

- [1] Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. Instructzero: Efficient instruction optimization for black-box large language models. In *ICML*, 2024.
- [2] Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Use your instinct: Instruction optimization for llms using neural bandits coupled with transformers. In *ICML*, 2024.
- [3] Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. Localized zeroth-order prompt optimization. *NeurIPS*, 2024.
- [4] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *EMNLP*, 2023.
- [5] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *ICLR*, 2024.
- [6] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*, 2024.
- [7] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *ICML*, 2020.
- [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*, 2024.
- [9] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mistral 7b. *arXiv*, 2023.
- [10] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [11] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Sch  lkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 2012.
- [12] Michael Arbel, Anna Korba, Adil Salim, and Arthur Gretton. Maximum mean discrepancy gradient flow. *Advances in Neural Information Processing Systems*, 2019.
- [13] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv*, 2015.
- [14] Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. Large sample analysis of the median heuristic. *arXiv*, 2017.