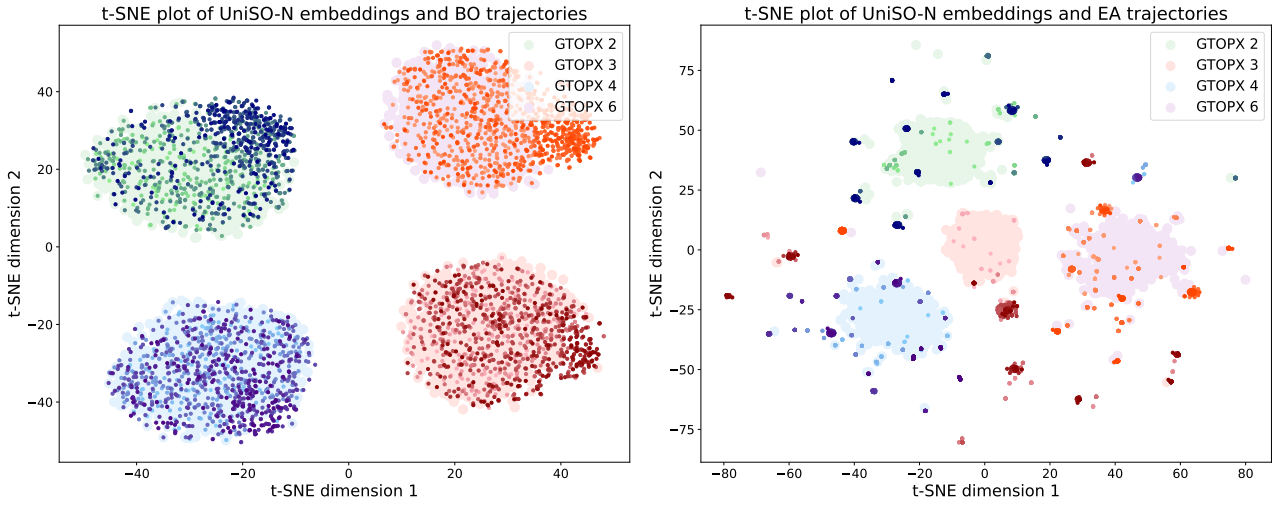

Figure S1. t-SNE plots of UniSO-N embedder comparing BO-qEI (left) and EA (right) search trajectories. Each part represents data from a single task, and small scatters represent the search trajectories of these two optimizers. Different colors represent different tasks, with darker colors indicating later stages of the search.

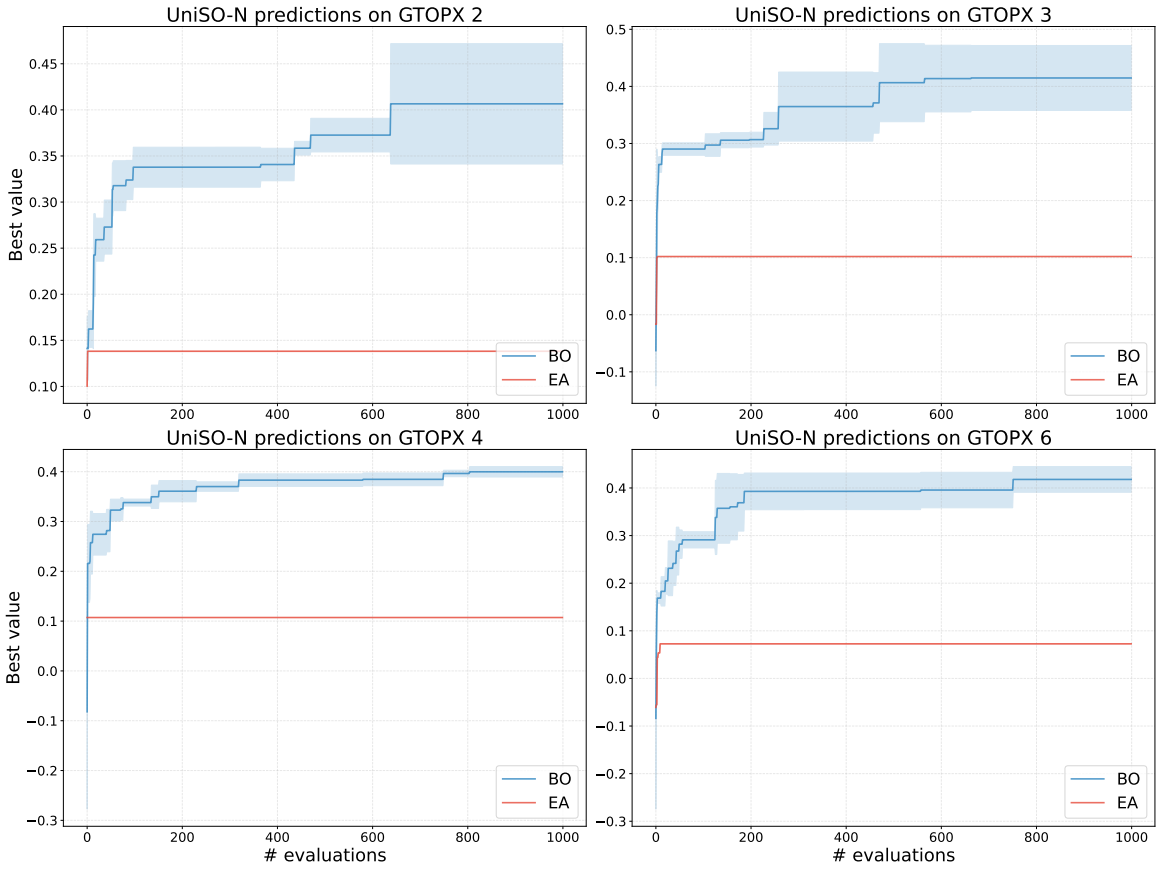

Figure S2. The historical best change prediction value curve of UniSO-N, with BO-qEI and EA as optimizers.

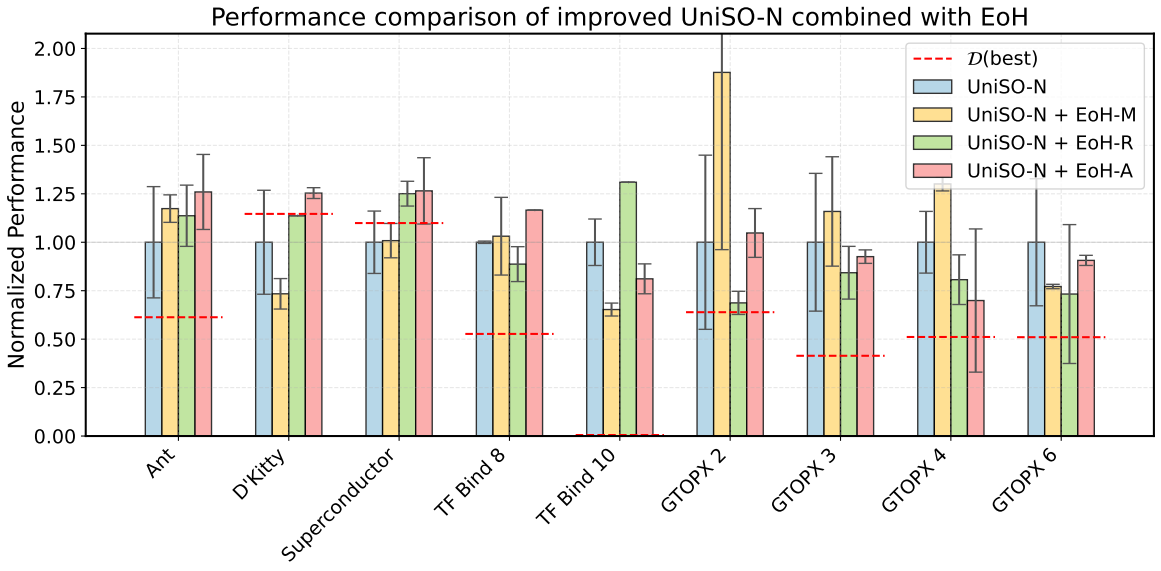

Figure S3. Performance comparison of improved UniSO-N combined with different EoH (Liu et al., 2024) strategies. The bar plot shows the normalized performance of the baseline UniSO-N model and three EoH-enhanced variants (EoH-M: metadata summarization, EoH-R: regularization-based fine-tuning, EoH-A: auxiliary loss implementation) across nine optimization tasks. The red dashed lines indicate the best score $\mathcal{D}$(best) in offline dataset for each task. Error bars represent standard deviations. The data is normalized by scaling UniSO-N's performance to 1 for each task.

Table S1. Un-normalized scores of improved UniSO-N and its variants with different EoH (Liu et al., 2024) strategies (EoH-M: metadata summarization, EoH-R: regularization-based fine-tuning, EoH-A: auxiliary loss implementation) in unconstrained tasks from Design-Bench and SOO-Bench, where the best and runner-up results on each task are **Blue** and **Violet**, respectively. $\mathcal{D}$(best) denotes the best score in the offline dataset.

Task	$\mathcal{D}$(best)	UniSO-N	UniSO-N + EoH-M	UniSO-N + EoH-R	UniSO-N + EoH-A
Ant	165.326	269.691 ± 77.425	**316.515 ± 19.141**	306.577 ± 42.614	**339.697 ± 52.210**
D'Kitty	**199.363**	173.911 ± 46.662	127.644 ± 13.676	197.777 ± 0.000	**218.026 ± 4.858**
Superconductor	74.000	67.333 ± 10.838	67.894 ± 5.964	**84.215 ± 4.306**	**85.186 ± 11.518**
TF Bind 8	0.439	0.833 ± 0.005	**0.859 ± 0.167**	0.739 ± 0.075	**0.971 ± 0.000**
TF Bind 10	0.005	**0.959 ± 0.115**	0.626 ± 0.032	**1.256 ± 0.000**	0.778 ± 0.074
GTOPX 2	-195.586	-124.995 ± 56.170	**-66.611 ± 32.466**	-181.919 ± 15.818	**-119.284 ± 14.325**
GTOPX 3	-151.190	**-62.622 ± 22.261**	**-54.032 ± 13.158**	-74.310 ± 11.999	-67.640 ± 2.542
GTOPX 4	-215.716	**-110.284 ± 17.559**	**-84.778 ± 2.349**	-136.679 ± 21.707	-157.717 ± 83.337
GTOPX 6	-112.599	**-57.435 ± 18.832**	-74.435 ± 1.096	-78.385 ± 38.336	**-63.365 ± 1.828**
Avg. Rank	/	2.667 ± 0.943	**2.333 ± 1.155**	3.000 ± 1.054	**2.000 ± 1.054**

Prompt for Initialization

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use LM embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. I need help generating metadata for 9 optimization tasks involving: robot morphology (Ant Morphology and D'Kitty Morphology), material science (Superconductor), DNA sequence optimization (TF Bind 8 and 10), and space mission trajectory optimization (GTOPX 2 3 4 6).

tasks information:
Ant Morphology and D'Kitty Morphology: robot morphology optimization. We created these two tasks to optimize the morphological structure of two simulated robots: Ant from OpenAI Gym and D'Kitty from ROBEL. For Ant Morphology, the goal is to optimize the morphology of an ant-shaped robot, to run as fast as possible, with a pre-trained neural network controller. For DKitty Morphology, the goal is to optimize the morphology of D'Kitty, a quadrupedal robot, such that a pre-trained neural network controller can navigate the robot to a fixed location. In this fashion, the goal for both tasks is to recover a morphology with which the pre-trained controller is compatible. The variable morphology parameters of both robots include size, orientation, and location of the limbs, giving us 60 continuous values in total for Ant and 56 for D'Kitty. To evaluate the ground-truth value for a given design, we run robotic simulation in MuJoCo for 100 timesteps, averaging 16 independent trials. These parameters are chosen to reduce stochasticity and allow the simulator to run in a minimal amount of time.
...

For each task, you need to generate 3 versions of metadata, where each metadata must be preceded by the task name. Do not generate additional explanations.

Prompt for E2

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use LM embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. I need help generating metadata for 9 optimization tasks involving: robot morphology (AntMorphology-v0 and DKittyMorphology-v0), material science (Superconductor-v0), DNA sequence optimization (TF Bind 8 and 10), and space mission trajectory optimization (GTOPX 2 3 4 6).

tasks information:
...

I have five existing metadatas as follows:
No.1 :
Code:
...
No.3 :
Code:

Please help me design a new algorithm that is different from the given ones but can be motivated by them.
Firstly, identify the common backbone idea in the provided metadatas.
Secondly, based on the backbone idea create your new metadatas.
You need to generate 9 metadata for the 9 tasks above respectively, each metadata must be preceded by the task name, and do not generate additional explanations.

Figure S4. Two examples of prompt engineering used in initialization and E2 strategy for EoH-M.

Prompt for E1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use LM embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. I need help generating metadata for 9 optimization tasks involving: robot morphology (Ant Morphology and D'Kitty Morphology), material science (Superconductor), DNA sequence optimization (TF Bind 8 and 10), and space mission trajectory optimization (GTOPX 2 3 4 6).

tasks information:
...
I have 3 existing metadatas as follows:
...

Please help me create new metadatas that has a totally different form from the given ones. You need to generate 9 metadata for the 9 tasks above respectively, each metadata must be preceded by the task name, and do not generate additional explanations.

Prompt for M1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,... }". I use LM embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. I need help generating metadata for 9 optimization tasks involving: robot morphology (Ant Morphology and D'Kitty Morphology), material science (Superconductor), DNA sequence optimization (TF Bind 8 and 10), and space mission trajectory optimization (GTOPX 2 3 4 6).

tasks information:
...

I have metadatas as follows:
No.1 :
Code:
...
No.3 :
Code:

Please assist me in creating new metadatas that has a different form but can be a modified version of the metadatas provided.
You need to generate 9 metadata for the 9 tasks above respectively, each metadata must be preceded by the task name, and do not generate additional explanations.

Figure S5. Two examples of prompt engineering used in E1 and M1 strategy for EoH-M.

Prompt for M3

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use LM embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. I need help generating metadata for 9 optimization tasks involving: robot morphology (Ant Morphology and D'Kitty Morphology), material science (Superconductor), DNA sequence optimization (TF Bind 8 and 10), and space mission trajectory optimization (GTOPX 2 3 4 6).

tasks information:
...

First, you need to identify the main components in the metadatas below:
No.1 :
Code:
...
No.3 :
Code:

Next, analyze whether any of these components can be overfit to the in-distribution instances.
Then, based on your analysis, simplify the components to enhance the generalization to potential out-of-distribution instances.
Finally, provide the revised metadata,, you need to generate 9 metadata for the 9 tasks above respectively, each metadata must be preceded by the task name, and do not generate additional explanations.

Figure S6. Two examples of prompt engineering used in M3 strategy for EoH-M. Note that we do not have an M2 strategy, since M2 in EoH (Liu et al., 2024) represents hyper-parameter adaptation while metadata do not have any hyper-parameters.

Prompt for Initialization

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.
Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named latent_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for E2

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

I have five existing algorithms with their codes as follows:
No.1 algorithms description:
Code:
...
No.3 algorithms description:
Code:

Please help me design a new algorithm that is different from the given ones but can be motivated by them.
Firstly, identify the common backbone idea in the provided algorithms.
Secondly, based on the backbone idea describe your new algorithm in one sentence.
Thirdly, implement it in Python as a function named 'latent_loss'. This function should accept two inputs: 'x_embedding', 'meta_embedding . The function should return one output: 'loss'. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D.The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S7. Two examples of prompt engineering used in initialization and E2 strategy for EoH-R.

Prompt for E1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

I need to design an additional loss function to further fine-tune the embedder checkpoint. The goal is to obtain better performances of the final solutions over all tasks.

Please help me create a new algorithm that has a totally different form from the given ones.

Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additional_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for M1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

I have one algorithm with its code as follows.
algorithms description:
Code:

Please assist me in creating a new algorithm that has a different form but can be a modified version of the algorithm provided.
Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additionalloss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S8. Two examples of prompt engineering used in E1 and M1 strategy for EoH-R.

Prompt for M2

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

I need to design an additional loss function to further fine-tune the embedder checkpoint. The goal is to obtain better performances of the final solutions over all tasks.

Please identify the main algorithm parameters and assist me in creating a new algorithm that has a different parameter settings of the score function provided.

Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named latent_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for M3

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I need to design a novel loss function to optimize the embedding representations of different tasks in the language model's latent space. The goal is to form a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks.

First, you need to identify the main components in the algorithm below:
algorithms description:
Code:

Next, analyze whether any of these components can be overfit to the in-distribution instances. **Then**, based on your analysis, simplify the components to enhance the generalization to potential out-of-distribution instances. **Finally**, provide the revised code, keeping the function name, inputs, and outputs unchanged.
Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S9. Two examples of prompt engineering used in M2 and M3 strategy for EoH-R.

Prompt for Initialization

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

I need to design an additional loss function to further fine-tune the embedder checkpoint. The goal is to obtain better performances of the final solutions over all tasks.
Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additional_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for E2

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

I have five existing algorithms with their codes as follows:
No.1 algorithms description:
Code:
...
No.3 algorithms description:
Code:

Please help me design a new algorithm that is different from the given ones but can be motivated by them.
Firstly, identify the common backbone idea in the provided algorithms.
Secondly, based on the backbone idea describe your new algorithm in one sentence.
Thirdly, implement it in Python as a function named 'additional_loss'. This function should accept two inputs: 'x_embedding', 'meta_embedding . The function should return one output: 'loss'. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D.The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S10. Two examples of prompt engineering used in initialization and E2 strategy for EoH-A.

Prompt for E1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

I need to design an additional loss function to further fine-tune the embedder checkpoint. The goal is to obtain better performances of the final solutions over all tasks.

Please help me create a new algorithm that has a totally different form from the given ones.

Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additional_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for M1

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

I have one algorithm with its code as follows.
algorithms description:
Code:

Please assist me in creating a new algorithm that has a different form but can be a modified version of the algorithm provided.
Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additional_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S11. Two examples of prompt engineering used in E1 and M1 strategy for EoH-A.

Prompt for M2

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

I need to design an additional loss function to further fine-tune the embedder checkpoint. The goal is to obtain better performances of the final solutions over all tasks.

Please identify the main algorithm parameters and assist me in creating a new algorithm that has a different parameter settings of the score function provided.

Firstly, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Python as a function named additional_loss. This function should accept two input(s): 'x_embedding', 'meta_embedding'. The function should return one output(s): loss. 'x_embedding' represents a batch of embeddings to be optimized with shape B*D (batch size * embedding dimension). 'meta_embedding' represents metadata information also provided in a batch embedding format with shape B*D. The output 'loss' is the designed loss value. Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Prompt for M3

I am solving universal offline black-box optimization (BBO) problem, i.e., solving multiple offline BBO task simultaneously. My approach is based on string-based representation of x by "{x1:**,x2:**,...}". I use a language model embedder to map the input string to an embedding space, and then apply an MLP head for regressing the objective. However, the LM embedder contains improper bias for regression. I have fine-tuned a T5-small embedder checkpoint that forms a clear distributional structure of embeddings in the latent space while maintaining distinguishability between tasks via the following implementation:

Loss code in our implementation...

First, you need to identify the main components in the algorithm below:
algorithms description:
Code:

Next, analyze whether any of these components can be overfit to the in-distribution instances. **Then**, based on your analysis, simplify the components to enhance the generalization to potential out-of-distribution instances. **Finally**, provide the revised code, keeping the function name, inputs, and outputs unchanged.
Note that both 'x_embedding' and 'meta_embedding' are torch tensors with matching batch sizes. The novel loss function should be sufficiently complex to achieve effective embedding optimization while maintaining computational stability. It is important to ensure the loss computation is mathematically sound and properly scaled.

All inputs and outputs are torch.Tensor. Do not give additional explanations.

Figure S12. Two examples of prompt engineering used in M2 and M3 strategy for EoH-A.

```
# Ant Morphology
{"type": "robot_design", "dimensions": 60, "input_range": [-1.0, 1.0], "output_range":
    [0, inf], "optimization_target": "running_speed", "simulation_engine": "MuJoCo", "
    evaluation_trials": 16, "simulation_length": 100}

# D'Kitty Morphology
{"type": "robot_design", "dimensions": 56, "input_range": [-1.0, 1.0], "output_range":
    [0, inf], "optimization_target": "target_reaching", "simulation_engine": "MuJoCo", "
    evaluation_trials": 16, "simulation_length": 100}

# Superconductor
{"type": "materials_science", "dimensions": 81, "input_range": [0.0, 1.0], "output_range"
    : [0, inf], "optimization_target": "critical_temperature", "model_type": "
    random_forest", "dataset_examples": 21263}

# TF Bind 8
{"type": "dna_design", "sequence_length": 8, "alphabet": ["A", "C", "G", "T"], "
    output_range": [0, 1], "optimization_target": "binding_affinity", "search_space":
    65792, "training_examples": 32898}

# TF Bind 10
{"type": "dna_design", "sequence_length": 10, "alphabet": ["A", "C", "G", "T"], "
    output_range": [0, 1], "optimization_target": "binding_affinity", "search_space":
    1048576, "training_examples": 50000}

# GTOPX 2
{"type": "spacecraft_trajectory", "dimensions": 22, "input_range": [-1.0, 1.0], "
    output_range": [0, inf], "optimization_target": "delta_v", "mission_type": "
    saturn_rendezvous", "maneuver": "dsm"}

# GTOPX 3
{"type": "spacecraft_trajectory", "dimensions": 18, "input_range": [-1.0, 1.0], "
    output_range": [0, inf], "optimization_target": "delta_v", "mission_type": "
    mercury_transfer", "flyby_type": "non_resonant"}

# GTOPX 4
{"type": "spacecraft_trajectory", "dimensions": 26, "input_range": [-1.0, 1.0], "
    output_range": [0, inf], "optimization_target": "delta_v", "mission_type": "
    mercury_transfer", "flyby_type": "resonant"}

# GTOPX 6
{"type": "spacecraft_trajectory", "dimensions": 22, "input_range": [-1.0, 1.0], "
    output_range": [0, inf], "optimization_target": "delta_v", "mission_type": "
    comet_rendezvous", "maneuver": "multi_gravity_assist"}
```

Figure S13. Best metadata found by EoH-M.

```python
import torch
import torch.nn.functional as F

def latent_loss(x_embedding: torch.Tensor, meta_embedding: torch.Tensor) -> torch.Tensor:
    batch_size = x_embedding.shape[0]
    # Multi-scale kernel density estimation
    scales = torch.tensor([0.1, 0.5, 1.0], device=x_embedding.device)
    dist_matrix = torch.cdist(x_embedding, x_embedding)

    density_loss = 0
    for scale in scales:
        kernel = torch.exp(-dist_matrix / (2 * scale * scale))
        density = kernel.sum(dim=1) / batch_size
        density_loss += -torch.log(density + 1e-6).mean()

    # Diffusion distance mapping
    transition = kernel / kernel.sum(dim=1, keepdim=True)
    diffused_x = torch.matmul(transition, x_embedding)
    diffusion_loss = torch.norm(diffused_x - x_embedding, dim=1).mean()

    # Task-specific repulsion
    x_norm = F.normalize(x_embedding, dim=1)
    meta_norm = F.normalize(meta_embedding, dim=1)
    task_sim = torch.matmul(x_norm, meta_norm.t())
    task_probs = F.softmax(task_sim / 0.1, dim=1)

    task_repulsion = torch.matmul(task_probs, task_probs.t())
    repulsion_loss = torch.mean(task_repulsion * torch.exp(-dist_matrix))

    # Smooth boundary transitions
    boundary_kernel = torch.exp(-task_sim)
    boundary_density = torch.matmul(boundary_kernel, task_probs)
    smoothness_loss = torch.mean((boundary_density - task_probs)**2)

    # Combined loss with balanced weights
    loss = (0.3 * density_loss +
            0.3 * diffusion_loss +
            0.2 * repulsion_loss +
            0.2 * smoothness_loss)

    return loss
```

Figure S14. Best code found by EoH-R. Description by the model: the algorithm extends the local structure preservation concept by incorporating adaptive kernel-based density estimation with multi-scale diffusion distances, while utilizing task-specific repulsion forces and smooth boundary transitions through probabilistic assignment matrices.

```python
import torch

def additional_loss(x_embedding: torch.Tensor, meta_embedding: torch.Tensor) -> torch.Tensor:
    x_dist = torch.cdist(x_embedding, x_embedding)
    m_dist = torch.cdist(meta_embedding, meta_embedding)

    x_rank = torch.argsort(torch.argsort(x_dist, dim=1), dim=1).float()
    m_rank = torch.argsort(torch.argsort(m_dist, dim=1), dim=1).float()

    rank_corr = torch.mean((x_rank - m_rank)**2)

    dist_ratio = x_dist / (m_dist + 1e-8)
    consistency = torch.mean(torch.abs(dist_ratio - torch.median(dist_ratio)))

    return rank_corr + 0.2 * consistency
```

Figure S15. Best code of EoH-A. Description by the model: balance pairwise distances in both embedding and metadata spaces while enforcing consistency through rank correlation.

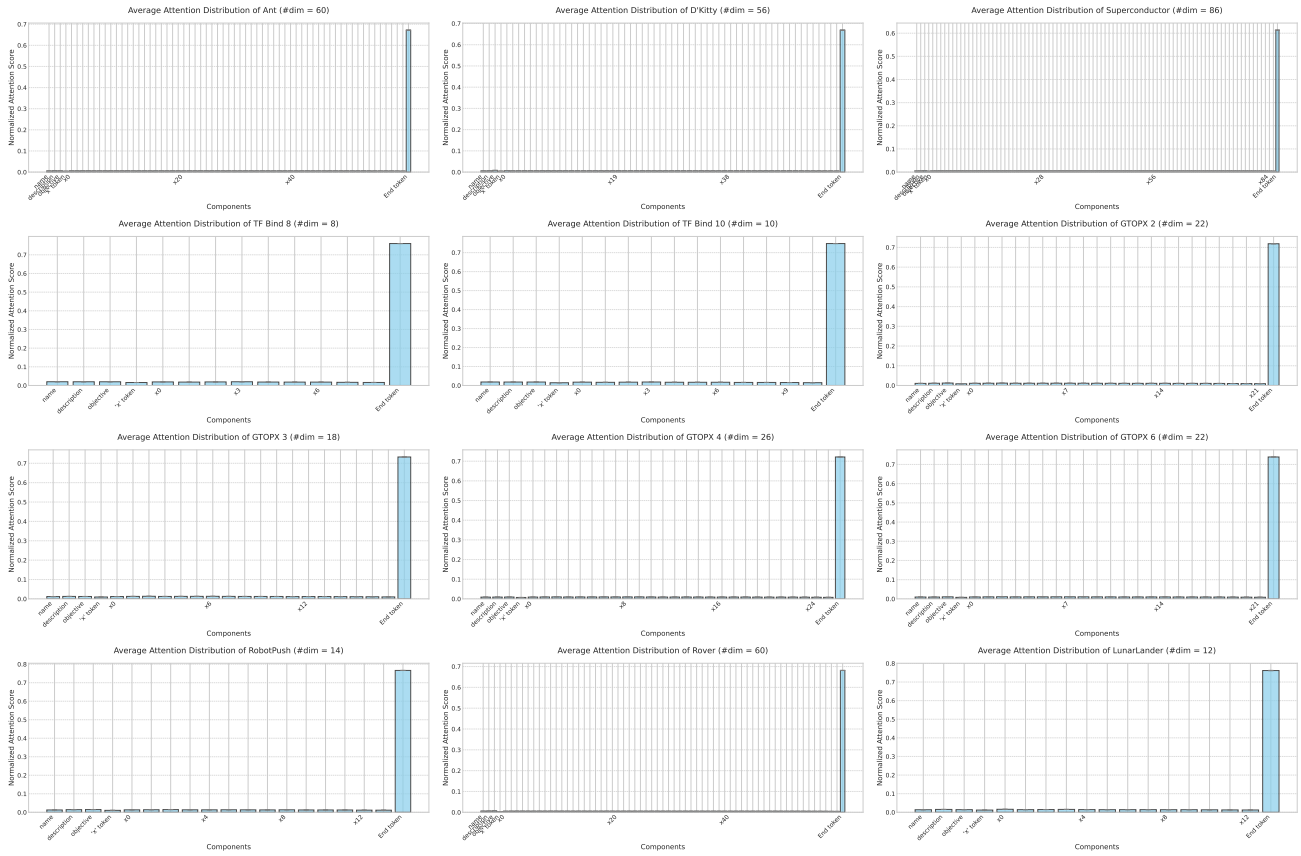

Figure S16. Average attention distribution (Allen-Zhu & Li, 2023) of pre-trained T5-small embedder on data from different tasks. The plots present attention patterns for 12 distinct tasks: Ant, D'Kitty, Superconductor, TF Bind 8, TF Bind 10, GTOPX 2, GTOPX 3 , GTOPX 4 , GTOPX 6, RobotPush, Rover, and LunarLander, which consistently show that the pre-trained model exhibits strong attention bias towards the `EOS` token across all tasks, demonstrating the model's focus on structural elements rather than numeric-solution-relevant components.

Figure S17. Average attention distribution (Allen-Zhu & Li, 2023) of T5-small embedder trained from scratch on different tasks. The plots show attention patterns across various components for 12 different tasks: Ant, D'Kitty, Superconductor, TF Bind 8, TF Bind 10, GTOPX 2, GTOPX 3, GTOPX 4, GTOPX 5, RobotPush, Rover, and LunarLander. The consistent pattern across tasks shows more balanced attention distribution on numeric-solution-relevant components, contrasting with pre-trained models' biases towards language-based structural tokens.

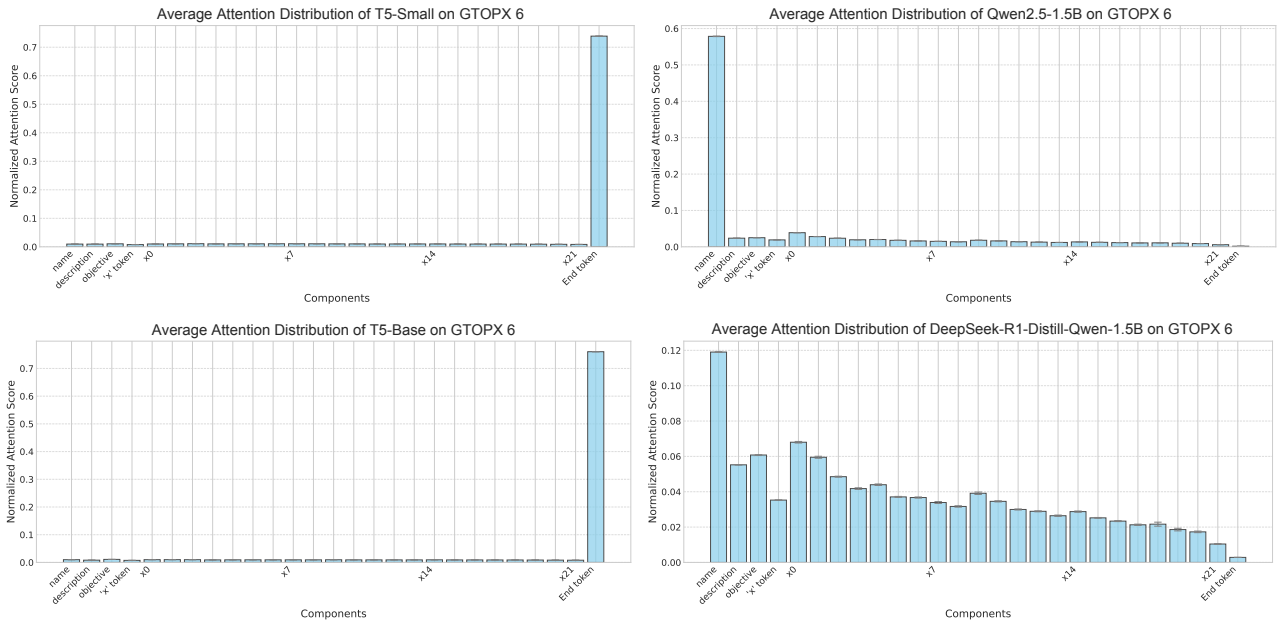

Figure S18. Different language models' average attention distribution (Allen-Zhu & Li, 2023) on GTOPX 6 task. The bar plots compare the normalized attention scores across different components for four models: T5-Small (top left), Qwen2.5-1.5B (top right), T5-Base (bottom left), and DeepSeek-R1-Distil-Qwen-1.5B (bottom right). Here, pre-trained T5 models exhibit strong attention bias towards the `EOS` token, while the DeepSeek-R1-Distill-Qwen-1.5B shows a more balanced attention distribution across numeric-solution-relevant components compared to Qwen2.5-1.5B, demonstrating the effectiveness of knowledge transfer in optimization tasks.

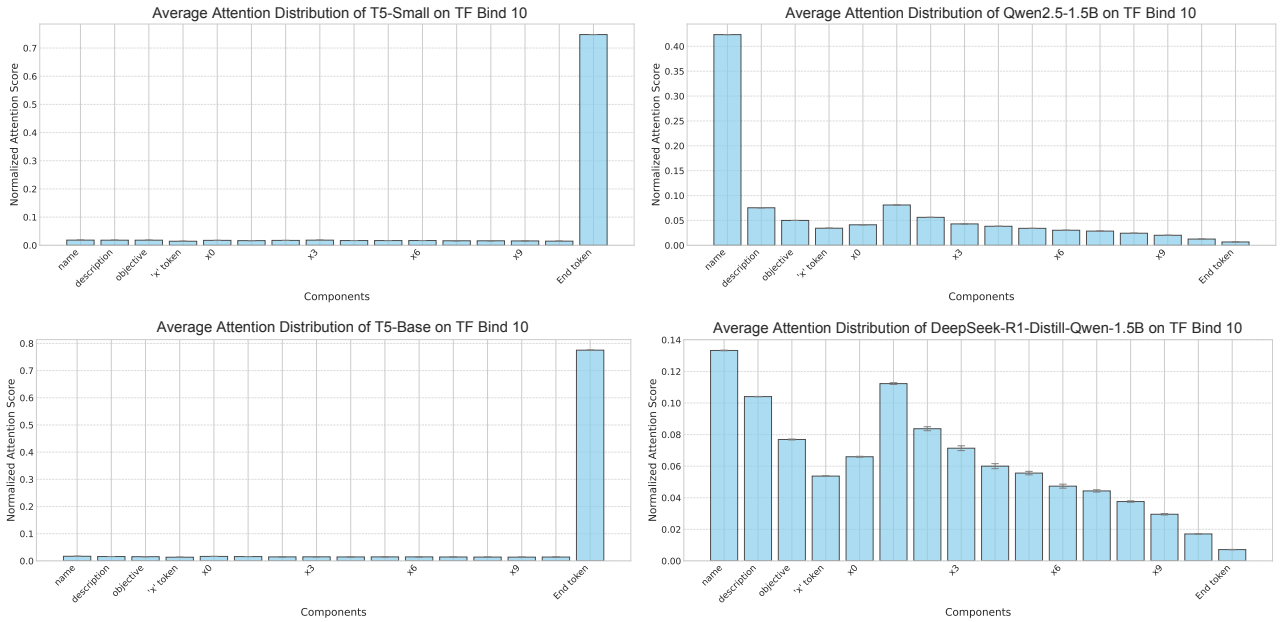

Figure S19. Different language models' average attention distribution (Allen-Zhu & Li, 2023) on GTOPX 6 task. The bar plots compare the normalized attention scores across different components for four models: T5-Small (top left), Qwen2.5-1.5B (top right), T5-Base (bottom left), and DeepSeek-R1-Distil-Qwen-1.5B (bottom right). Similar to that in GTOPX 6, pre-trained T5 models exhibit strong attention bias towards the `EOS` token, while the DeepSeek-R1-Distill-Qwen-1.5B shows a more balanced attention distribution across numeric-solution-relevant components compared to Qwen2.5-1.5B, demonstrating the effectiveness of knowledge transfer in optimization tasks.

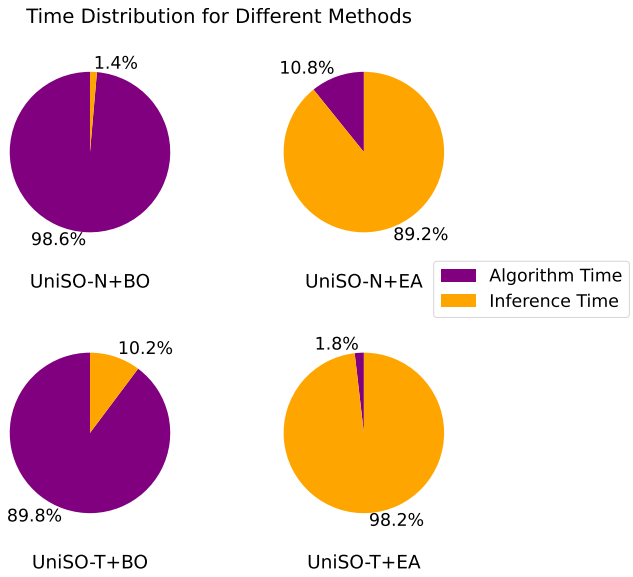

Figure S20. Time distribution comparison between algorithm time and model inference time for different UniSO models (UniSO-N, UniSO-T) and optimization methods (BO-qEI, EA). The pie charts show the percentage breakdown between algorithm time (purple) and inference time (orange) for each model.

Table S2. Computational budgets of improved UniSO-N and UniSO-T with different black-box optimizers (BO-qEI, EA) on unconstrained tasks from Design-Bench and SOO-Bench. Each cell shows algorithm time / model inference time (in seconds). Here, BO-qEI and EA are allowed for an evaluation budget of 1000 (i.e., the optimizer can access the model's outputs for upmost 1000 times). EA-25600 denotes the EA with an extended evaluation budget of 25600, implemented with a population size of 128 over 200 generations, which is our default optimizer setting in the submitted version.

Model	UniSO-N			UniSO-T		
Search method	BO-qEI	EA	EA-25600	BO-qEI	EA	EA-25600
Ant	74.44±0.26 / 1.87±0.00	0.25±0.00 / 1.94±0.00	2.75±0.02 / 38.16±0.00	74.06±0.27 / 14.81±0.00	0.26±0.00 / 14.55±0.00	2.62±0.00 / 202.05±0.06
D'Kitty	78.22±1.03 / 1.72±0.00	0.23±0.00 / 2.00±0.00	2.59±0.00 / 37.75±0.00	77.14±0.98 / 13.75±0.01	0.26±0.00 / 13.94±0.01	2.59±0.00 / 196.90±0.02
Superconductor	73.48±0.00 / 1.89±0.00	0.23±0.00 / 1.92±0.00	2.68±0.00 / 40.60±0.03	73.78±0.00 / 14.86±0.01	0.27±0.00 / 14.74±0.01	2.80±0.01 / 214.94±0.01
TF Bind 8	59.67±0.00 / 1.58±0.00	0.27±0.00 / 1.77±0.00	3.08±0.01 / 35.02±0.00	60.47±0.00 / 13.02±0.06	0.26±0.00 / 13.17±0.07	2.46±0.01 / 170.63±0.00
TF Bind 10	62.57±0.02 / 1.59±0.00	0.21±0.00 / 1.76±0.00	2.76±0.01 / 35.28±0.05	62.08±0.02 / 13.03±0.03	0.25±0.00 / 13.01±0.03	2.33±0.00 / 170.98±0.15
GTOPX 2	181.67±8.44 / 1.71±0.00	0.21±0.00 / 1.94±0.00	2.34±0.00 / 35.20±0.01	182.48±8.26 / 13.46±0.00	0.25±0.00 / 13.66±0.00	2.49±0.02 / 182.35±0.48
GTOPX 3	179.39±6.39 / 1.58±0.00	0.22±0.00 / 1.79±0.00	2.53±0.00 / 34.82±0.00	179.07±6.20 / 13.23±0.00	0.24±0.00 / 13.08±0.00	2.45±0.02 / 178.54±0.07
GTOPX 4	189.18±39.94 / 1.62±0.00	0.21±0.00 / 1.84±0.00	2.37±0.00 / 35.75±0.01	189.88±40.37 / 13.04±0.00	0.25±0.00 / 13.28±0.00	2.49±0.01 / 184.38±0.00
GTOPX 6	180.05±17.91 / 1.60±0.00	0.22±0.00 / 1.92±0.00	2.37±0.00 / 35.98±0.00	176.91±18.75 / 13.50±0.00	0.25±0.00 / 13.26±0.00	2.56±0.00 / 182.37±0.10