

# Hyperparameter Tuning for Milvus via HOB0

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Method</b>	<b>2</b>
2.1	Loss Function . . . . .	3
<b>3</b>	<b>Implementation</b>	<b>3</b>
3.1	Hardware Information . . . . .	3
3.2	Core Code Annotations . . . . .	3
3.2.1	ENV . . . . .	3
3.2.2	Search Config Class . . . . .	4
3.2.3	ENV Input . . . . .	4
3.3	User Input Parser . . . . .	5
3.4	HOBO . . . . .	6
<b>4</b>	<b>Results</b>	<b>7</b>
<b>5</b>	<b>Attention</b>	<b>9</b>
5.1	VAE . . . . .	9
5.2	Graph Based Message Passing . . . . .	9
5.3	Minimum Distortion Embedding . . . . .	9

# 1 Introduction

For solving the optimization of milvus hyperparameters, we use the Bayesian Optimization and Hyperband(BOHB)[2] as our parameter search method.

## 2 Method

There are several level hyperparameters in milvus, including index-type, index-params and search-params. To get an end-to-end solution for index, we use BOHB in different level. For Index Type, to eminent the randomness of BO for index-type(which means BO may not fully explore some specific type due to init poor performance), we set two index type optimization mode(Loop and BO).

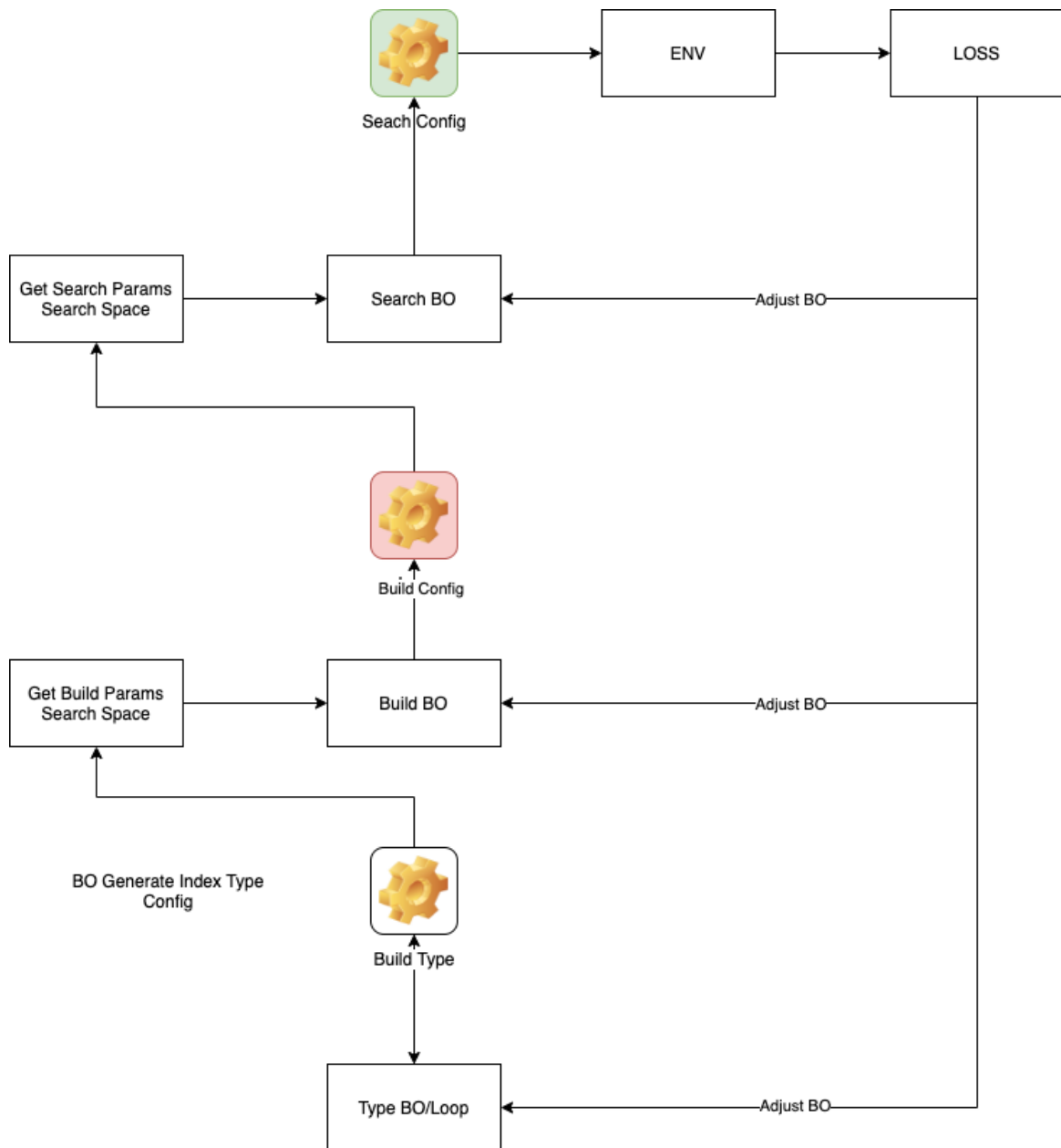


Figure 1: Flow Diagram

## 2.1 Loss Function

We use laplace method to convert constraint BO to unconstraint version. Our loss function is as below:

$$Loss = sign(recall, threshold) - query\_per\_sec \quad (1)$$

$$Sign(recall, threshold) = \begin{cases} recall - threshold & recall > threshold \\ \lambda * (threshold - x) & recall \leq threshold, \end{cases} \quad (2)$$

Here we use  $\lambda = 100000$  for Lagrange method, and  $threshold = 95$ .

## 3 Implemtation

### 3.1 Hardware Information

Our method is tested on server with: CPU: Intel Core i7-8700 CPU @ 4.6GHz and RAM: 32083MiB.

### 3.2 Core Code Annotations

#### 3.2.1 ENV

ENV is a helper class configure basic milvs related variables.

Listing 1: ENV class config

```
1 class ENV():
2 def __init__(self, args = None):
3     print("ENV")
4     # docker related information
5     host = '127.0.0.1'
6     port = '19530'
7
8     # get milvus client and collection_name
9     self.client = Milvus(host, port)
10    self.collection_name = args.collection_name
11
12    # get query_vectors and set top_k
13    self.query_groundtruth = self.get_groundtruth()
14    self.query_vectors = self.get_query()
15    self.top_k = 100
16
17    # get status by curretn db
18    self.index_type = None
19    self.index_params = None
20    self.refresh_status()
21
22    # set datadim, which is needed by some serch constraint
23    global gDataDim
24    gDataDim = 128
25
26    # based on the input type, get the default build config
27    if args.op == "build_params":
```

```

28     self.target_index_type = get_index_type(args.index_type)
29     self.target_index_params = \
30     get_default_build_config(self.target_index_type)
31
32     is_build = False
33     if self.index_type != self.target_index_type:
34         is_build = True
35     elif self.index_params != self.target_index_params:
36         is_build = True
37
38     if is_build:
39         self.env_build_input(self.target_index_type, \
40                             self.target_index_params)
41         self.refresh_status()
42
43     # set search space
44     self.default_build_config = get_default_build_config(self.index_type)
45     self.search_configspace = get_search_configspace(self.index_type, \
46                                                       self.index_params)
47     self.build_configspace = get_build_configspace(self.index_type)

```

Listing 2: Refresh Status

```

1  # base on user's input, get the target search space
2  def refresh_status(self):
3      """
4      refresh status
5      reset index_type and index_params
6      reset all config space
7      """
8      status, stats = self.client.get_index_info(self.collection_name)
9      self.index_type = stats._index_type
10     self.index_params = stats._params
11
12     # set config space
13     self.default_build_config = get_default_build_config(self.index_type)
14     self.build_configspace = get_build_configspace(self.index_type)
15     self.search_configspace = get_search_configspace(self.index_type, \
16                                                       self.index_params)

```

### 3.2.2 Search Config Class

Listing 3: Search Config Class

```

1  class HNSW_build_search_shared_config(object):
2  def __init__(self):
3      self.M = cs.IntegerUniformHyperparameter('M', 4, 64)
4      self.efConstruction = \
5          cs.IntegerUniformHyperparameter('efConstruction', 8, 512)
6      top_k = 100
7      self.ef = cs.IntegerUniformHyperparameter('ef', top_k, 512)
8      self.configspace = \
9          cs.ConfigurationSpace([self.M, self.efConstruction, self.ef], seed=123)

```

We use constant class to configure the default search space.

### 3.2.3 ENV Input

Listing 4: Input

```

1 # given full env put
2 def config_input(self, config):
3     # check current index type
4     is_build = False
5
6     self.refresh_status()
7     if self.index_type != config['index_type']:
8         is_build = True
9     elif self.index_params != config['index_params']:
10        is_build = True
11
12    if is_build:
13        self.env_build_input(config['index_type'], config['index_params'])
14        self.refresh_status()
15
16
17    recall, query_per_sec = self.env_search_input(config['search_params'])
18    self.search_params = config['search_params']
19
20    return recall, query_per_sec

```

For input, we check the current status and mark the change that need to be done. Once we have changed the index of current database, we refresh the ENV status.

### 3.3 User Input Parser

Listing 5: Input Parser

```

1 if args.op == "build_type":
2     build_type_search_spcas = \
3     [IndexType.IVF_FLAT, IndexType.IVF_PQ, IndexType.IVF_SQ8, IndexType.HNSW]
4     if args.build_type_op_method == "BO": # BO
5         index_type = \
6         cs.CategoricalHyperparameter('index_type', build_type_search_spcas)
7         index_type_configspace = cs.ConfigurationSpace([index_type], seed=123)
8         type_opt = \
9         BOHB(index_type_configspace, \
10            build_type_evaluate, max_budget=10, min_budget=1)
11        type_logs = type_opt.optimize()
12    else: # Loop
13        for index_type in build_type_search_spcas:
14            env.target_index_type = index_type
15            env.refresh_status()
16            opt = \
17            BOHB(get_build_configspace(env.target_index_type), \
18               build_evaluate, max_budget=10, min_budget=1)
19            logs = opt.optimize()

```

The input parser is trivial.

## 3.4 HOBO

Listing 6: HOBO

```
1 # Based on the HOBOb class and our search space configuration , \
2 # we can build a BOHB object. After that, we can call the optimize() \
3 # method to start the optimization process.
4 def build_type_evaluate(params, n_iterations):
5     env.target_index_type = params['index_type']
6     env.refresh_status()
7     if args.build_search_share_space:
8         opt = BOHB(get_build_search_shared_configspace(env.target_index_type),
9                    build_search_share_space_evaluate,
10                    max_budget=n_iterations,
11                    min_budget=1,
12                    eta = 10)
13         logs = opt.optimize()
14     else:
15         opt = BOHB(get_build_configspace(env.target_index_type),
16                    build_evaluate,
17                    max_budget=n_iterations,
18                    min_budget=1,
19                    eta = 10)
20         logs = opt.optimize()
21     return logs.best['loss']
```

# 4 Results



# HOBO Results

Set 25, 2021

## Results

### Index Type Optimization

Method	index_type	M	efConstruction	ef	recall	query_per_sec	loss
BOHB(Index Type Loop)	'HNSW'	17	445	114	99.68	16782.6	-16777.9
BOHB(Index Type BO)	'HNSW'	22	274	106	99.75	18522	-18517.2

	index_type	nlist	M	nprobe	recall	query_per_sec	loss
Grid Search	'HNSW'	4	158	200	97.11	18331.74	-18329.63

### Index Parameters Optimization

#### IVF\_FLAT

	index_type	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_FLAT'	2883	54	99.68	14911	-14906.3
Grid Search	'IVF_FLAT'	14601	101	100.0	14402.03	-14397.03

#### IVF\_SQ8

	index_type	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_SQ8'	8405	46	98.86	13827.5	-13823.7
Grid Search	'IVF_SQ8'	5401	101	99.49	13080.62	-13076.13

#### IVF\_PQ

	index_type	m	nlist	nprobe	recall	query_per_sec	loss
BOHB	'IVF_PQ'	128	3800	205	98.1	1289.00	-1285.90
Grid Search	'IVF_PQ'	64	1	1	95.08	1733.67	-7629.25

#### HNSW

Method	index_type	M	efConstruction	ef	recall	query_per_sec	loss
BOHB	'HNSW'	18	92	157	99.85	17868.6	-17863.8
Grid Search	'HNSW'	4	158	200	97.11	18331.74	-18329.63

## 5 Attention

### 5.1 VAE

Trying to compress the embedding dimension and get a more milvus-friendly representation space, we are using VAE to preprocess the given datasets.

#### Minimum Reconstruct Error

$$Loss(eb) \tag{3}$$

### 5.2 Graph Based Message Passing

### 5.3 Minimum Distortion Embedding

[1]

## References

- [1] Akshay Agrawal, Alnur Ali, and Stephen Boyd. Minimum-Distortion Embedding. page 179.
- [2] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. page 10.