A OVERALL ARCHITECTURE OF SHiFT

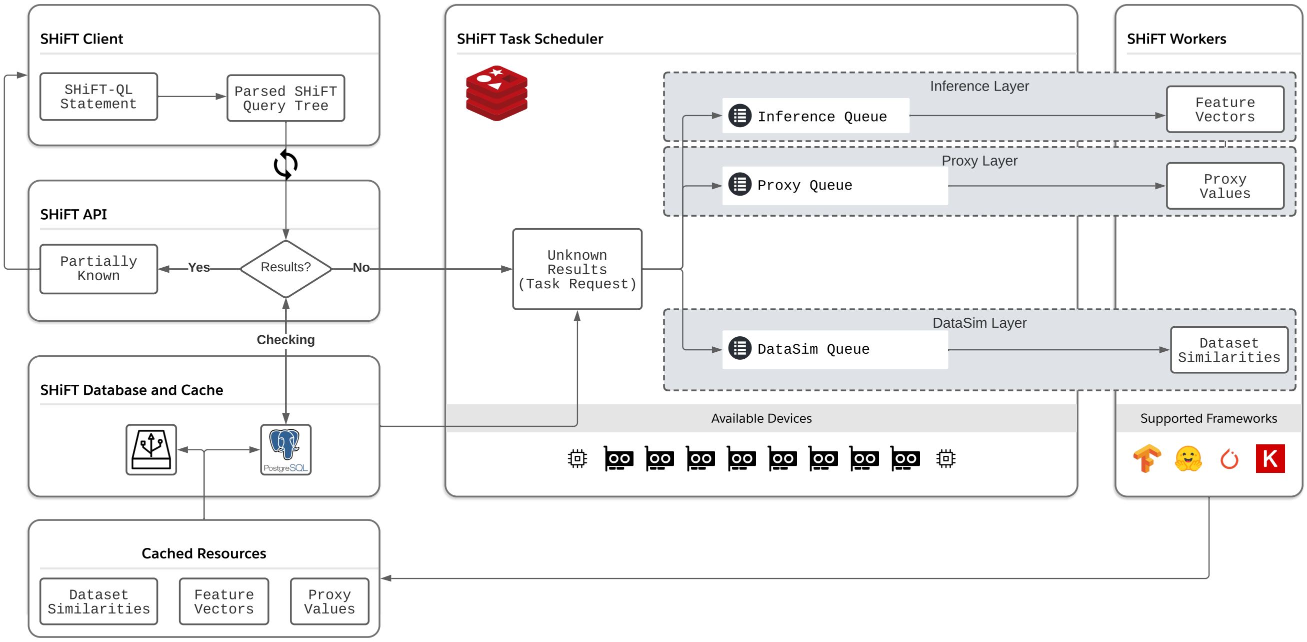

Figure 13: Overall system architecture of SHiFT.

B EXAMPLE SHIFT-QL QUERIES

Given the flexible logical abstraction of SHiFT, we are able to express a diverse range of search strategies, while allowing a user to conduct her own filtering and selections operations using standard SQL queries. As an example, the five popular search strategies in Table 1 can be expressed in SHiFT as follows:

```
Q1 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY UpstreamAccuracy DESC LIMIT 1
Q2 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY CosineNN ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader
Q3 := SELECT ModelId FROM Models
      WHERE Input = 'Vision'
      ORDER BY Linear(lr=0.1) ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader
Q4 := Q1
      UNION
      SELECT ModelId FROM Models
      WHERE Input = 'Vision' AND ModelId NOT IN Q1
      ORDER BY Linear(lr=0.1) ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader
Q5 := SELECT ModelId FROM Models
      WHERE Input = 'Vision' AND
            DataReaders.ReaderId IN
            (
                SELECT DataReaderId FROM DataReaders
                ORDER BY Task2Vec LIMIT 1
                TESTED ON TestReader
            ) Q6
      ORDER BY FineTune LIMIT 1
```

More Complex Nested Query. Assuming users know they have a *structured* computer vision dataset, it might be useful to prob only the best models for all structured datasets in the list of benchmark datasets. Noting that fine-tuning that many models can still be rather expensive, picking only the best task-aware model (e.g., via a linear proxy) out of those can further speedup the search process. This concrete example translates to the following SHiFT-QL query Q7, where Q8 represent the meta-learned task-agnostic part.

```
Q7 := SELECT ModelId FROM
      (
        SELECT ModelId FROM BenchmarkResults
        NATURAL JOIN DataReaders
        WHERE Model.Input = 'Vision'
          AND DataReader.Type == 'Structured'
        RETRIEVE 2 GRP DataReaderId
          ORD Accuracy DESC
      ) as Q8
      ORDER BY Linear(lr=0.1) ASC LIMIT 1
      TESTED ON TestReader TRAINED ON TrainReader
```

Notice that SHiFT uses a syntactical sugar RETRIEVE to enable users of SHiFT-QL to retrieve the top 2 elements grouped by the attribute DataReaderId ordered by the attribute Accuracy. Internally, this keyword is automatically translated to a standard SQL query using the statements PARTITION BY and RANK()[4].

[4] The implementation is PostgreSQL specific. The function and statement names might be different for other distributions.

C SUCCESSIVE-HALVING ALGORITHM

Algorithm 1 Successive-Halving [12]

```
Input: Budget $B$, chunk size $C$, $M$ candidate models with $l_{i,k}$ denoting the loss of the $i$th model trained on the data samples in $[0, k]$
Initialize: $S_0 = [M]$
for $k = 0, 1, \dots, \lceil \log_2(|M|) \rceil - 1$ do
    Let $L = |S_k|$;
    Evaluate each model $i = 1, \dots, L$ with $r_k \times C$ more training data samples where $r_k = \lfloor \frac{B}{L \cdot \lceil \log_2(|M|) \rceil} \rfloor$
    Set $R_k = C \times \sum_{j=0}^{k} r_j$
    Let $\sigma_k$ be a permutation on $S_k$ s.t. $l_{\sigma_k(1),R_k} \leq \dots \leq l_{\sigma_k(|S_k|),R_k}$
    Let $S_{k+1} = \{\sigma_k(1), \dots, \sigma_k(\lceil L/2 \rceil)\}$.
    if No data to perform more arm pulls then
        Output: $\sigma_k(1)$
    end if
end for
Output: Singleton element of $S_{\lceil \log_2(|M|) \rceil}$
```

D MODEL DETAILS

Table 3: All vision models are available with “https://tfhub.dev/” as prefix (part 1/2).

Model	Inference Cost (ms)
google/cropnet/feature_vector/cassava_disease_V1/1	11
google/cropnet/feature_vector/cassava_disease_V1/1	11
google/cropnet/feature_vector/concat/1	12
google/cropnet/feature_vector/imagenet/1	12
google/imagenet/efficientnet_v2_imagenet1k_b0/feature_vector/2	11
google/imagenet/efficientnet_v2_imagenet1k_b1/feature_vector/2	14
google/imagenet/efficientnet_v2_imagenet1k_b2/feature_vector/2	14
google/imagenet/efficientnet_v2_imagenet1k_b3/feature_vector/2	17
google/imagenet/efficientnet_v2_imagenet1k_l/feature_vector/2	85
google/imagenet/efficientnet_v2_imagenet1k_m/feature_vector/2	55
google/imagenet/efficientnet_v2_imagenet1k_s/feature_vector/2	23
google/imagenet/efficientnet_v2_imagenet21k_b0/feature_vector/2	11
google/imagenet/efficientnet_v2_imagenet21k_b1/feature_vector/2	11
google/imagenet/efficientnet_v2_imagenet21k_b2/feature_vector/2	12
google/imagenet/efficientnet_v2_imagenet21k_b3/feature_vector/2	14
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b0/feature_vector/2	11
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b1/feature_vector/2	12
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b2/feature_vector/2	13
google/imagenet/efficientnet_v2_imagenet21k_ft1k_b3/feature_vector/2	14
google/imagenet/efficientnet_v2_imagenet21k_ft1k_l/feature_vector/2	73
google/imagenet/efficientnet_v2_imagenet21k_ft1k_m/feature_vector/2	33
google/imagenet/efficientnet_v2_imagenet21k_ft1k_xl/feature_vector/2	74
google/imagenet/efficientnet_v2_imagenet21k_l/feature_vector/2	63
google/imagenet/efficientnet_v2_imagenet21k_m/feature_vector/2	33
google/imagenet/efficientnet_v2_imagenet21k_s/feature_vector/2	21
google/imagenet/inception_resnet_v2/feature_vector/4	26
google/imagenet/inception_v1/feature_vector/4	13
google/imagenet/inception_v2/feature_vector/4	11
google/imagenet/inception_v3/feature_vector/4	14
google/imagenet/inception_v3/feature_vector/5	18
google/imagenet/mobilenet_v1_025_128/feature_vector/5	6
google/imagenet/mobilenet_v1_025_160/feature_vector/5	6
google/imagenet/mobilenet_v1_025_192/feature_vector/5	6
google/imagenet/mobilenet_v1_025_224/feature_vector/5	6
google/imagenet/mobilenet_v1_050_128/feature_vector/5	6
google/imagenet/mobilenet_v1_050_160/feature_vector/5	8
google/imagenet/mobilenet_v1_050_192/feature_vector/5	6
google/imagenet/mobilenet_v1_050_224/feature_vector/5	6
google/imagenet/mobilenet_v1_075_128/feature_vector/5	7
google/imagenet/mobilenet_v1_075_160/feature_vector/5	6
google/imagenet/mobilenet_v1_075_192/feature_vector/5	6
google/imagenet/mobilenet_v1_075_224/feature_vector/5	6
google/imagenet/mobilenet_v1_100_128/feature_vector/5	6
google/imagenet/mobilenet_v1_100_160/feature_vector/5	7
google/imagenet/mobilenet_v1_100_192/feature_vector/5	6
google/imagenet/mobilenet_v1_100_224/feature_vector/4	7
google/imagenet/mobilenet_v2_035_128/feature_vector/5	8
google/imagenet/mobilenet_v2_035_160/feature_vector/5	8
google/imagenet/mobilenet_v2_035_192/feature_vector/5	9
google/imagenet/mobilenet_v2_035_224/feature_vector/5	8
google/imagenet/mobilenet_v2_035_96/feature_vector/5	9

Table 4: All vision models are available with “https://tfhub.dev/” as prefix (part 2/2).

Model	Inference Cost (ms)
google/imagenet/mobilenet_v2_050_128/feature_vector/5	9
google/imagenet/mobilenet_v2_050_160/feature_vector/5	8
google/imagenet/mobilenet_v2_050_192/feature_vector/5	8
google/imagenet/mobilenet_v2_050_224/feature_vector/5	9
google/imagenet/mobilenet_v2_050_96/feature_vector/5	8
google/imagenet/mobilenet_v2_075_128/feature_vector/5	9
google/imagenet/mobilenet_v2_075_160/feature_vector/5	11
google/imagenet/mobilenet_v2_075_192/feature_vector/5	11
google/imagenet/mobilenet_v2_075_224/feature_vector/5	10
google/imagenet/mobilenet_v2_075_96/feature_vector/5	11
google/imagenet/mobilenet_v2_100_128/feature_vector/5	9
google/imagenet/mobilenet_v2_100_160/feature_vector/5	9
google/imagenet/mobilenet_v2_100_192/feature_vector/5	11
google/imagenet/mobilenet_v2_100_224/feature_vector/4	9
google/imagenet/mobilenet_v2_100_96/feature_vector/5	8
google/imagenet/mobilenet_v2_130_224/feature_vector/5	9
google/imagenet/mobilenet_v2_140_224/feature_vector/5	11
google/imagenet/mobilenet_v3_large_075_224/feature_vector/5	11
google/imagenet/mobilenet_v3_large_100_224/feature_vector/5	10
google/imagenet/mobilenet_v3_small_075_224/feature_vector/5	9
google/imagenet/mobilenet_v3_small_100_224/feature_vector/5	9
google/imagenet/nasnet_mobile/feature_vector/4	19
google/imagenet/resnet_v1_101/feature_vector/4	16
google/imagenet/resnet_v1_152/feature_vector/4	22
google/imagenet/resnet_v1_50/feature_vector/4	13
google/imagenet/resnet_v2_101/feature_vector/4	16
google/imagenet/resnet_v2_152/feature_vector/4	22
google/imagenet/resnet_v2_50/feature_vector/4	13
tensorflow/efficientnet/b0/feature-vector/1	12
tensorflow/efficientnet/b1/feature-vector/1	15
tensorflow/efficientnet/b2/feature-vector/1	17
tensorflow/efficientnet/b3/feature-vector/1	21
tensorflow/efficientnet/b4/feature-vector/1	32
tensorflow/efficientnet/b5/feature-vector/1	44
tensorflow/efficientnet/b6/feature-vector/1	79
tensorflow/efficientnet/b7/feature-vector/1	151
vtab/exemplar/1	13
vtab/jigsaw/1	25
vtab/relative-patch-location/1	21
vtab/rotation/1	12
vtab/semi-exemplar-10/1	12
vtab/semi-rotation-10/1	12
vtab/sup-100/1	12
vtab/sup-exemplar-100/1	11
vtab/sup-rotation-100/1	12
vtab/uncond-biggan/1	17
vtab/vae/1	10
vtab/wae-gan/1	10
vtab/wae-mmd/1	10
vtab/wae-ukl/1	10

Table 5: All NLP models are available as HuggingFace transformers (part 1/2).

Model	Inference Cost (ms)
18811449050/bert_finetuning_test	18
aditeyabaral/finetuned-sail2017-xlm-roberta-base	17
aliosm/sha3bor-metre-detector-arabertv2-base	21
Alireza1044/albert-base-v2-qnli	22
anferico/bert-for-patents	59
anirudh21/bert-base-uncased-finetuned-qnli	18
ASCCCCCCCC/distilbert-base-chinese-amazon_zh_20000	21
aviator-neural/bert-base-uncased-sst2	19
aychang/bert-base-cased-trec-coarse	21
bert-base-cased	19
bert-base-uncased	19
bert-large-uncased	55
bondi/bert-semaphore-prediction-w4	21
CAMeL-Lab/bert-base-arabic-camelbert-da-sentiment	21
CAMeL-Lab/bert-base-arabic-camelbert-mix-did-nadi	21
Capreolus/bert-base-msmarco	17
chiragasarpota/scotus-bert	6
classla/bcms-bertic-parlasent-bcs-ter	21
connectivity/bert_ft_qqp-1	19
connectivity/bert_ft_qqp-17	19
connectivity/bert_ft_qqp-25	19
connectivity/bert_ft_qqp-7	21
connectivity/bert_ft_qqp-94	21
connectivity/bert_ft_qqp-96	19
connectivity/feather_berts_28	17
dhimskyy/wiki-bert	13
DoyyingFace/bert-asian-hate-tweets-asian-unclean-freeze-4	19
emrecan/bert-base-multilingual-cased-snli_tr	21
gchhablani/bert-base-cased-finetuned-rte	19
gchhablani/bert-base-cased-finetuned-wnli	18
Guscode/DKbert-hatespeech-detection	20
ishan/bert-base-uncased-mnli	17
jb2k/bert-base-multilingual-cased-language-detection	22
Jeevesh8/512seq_len_6ep_bert_ft_cola-91	17
Jeevesh8/6ep_bert_ft_cola-12	17
Jeevesh8/6ep_bert_ft_cola-29	18
Jeevesh8/6ep_bert_ft_cola-47	19

Table 6: All NLP models are available as HuggingFace transformers (part 2/2).

Model	Inference Cost (ms)
Jeevesh8/bert_ft_cola-60	19
Jeevesh8/bert_ft_cola-88	19
Jeevesh8/bert_ft_qqp-39	20
Jeevesh8/bert_ft_qqp-40	19
Jeevesh8/bert_ft_qqp-55	20
Jeevesh8/bert_ft_qqp-68	21
Jeevesh8/bert_ft_qqp-88	19
Jeevesh8/bert_ft_qqp-9	19
Jeevesh8/feather_berts_46	17
Jeevesh8/feather_berts_96	17
Jeevesh8/init_bert_ft_qqp-24	21
Jeevesh8/init_bert_ft_qqp-28	21
Jeevesh8/init_bert_ft_qqp-33	19
Jeevesh8/init_bert_ft_qqp-49	21
Jeevesh8/lecun_feather_berts-3	17
Jeevesh8/lecun_feather_berts-51	17
manueltonneau/bert-twitter-en-is-hired	19
Monsia/camembert-fr-covid-tweet-classification	21
moshew/bert-mini-sst2-distilled	2
mujeensung/bert-base-cased_mnli_bc	17
navsad/navid_test_bert	17
oferweintraub/bert-base-finance-sentiment-noisy-search	18
Recognai/bert-base-spanish-wwm-cased-xnli	21
socialmediaie/TRAC2020_IBEN_B_bert-base-multilingual-uncased	21
Splend1dchan/bert-base-uncased-slue-goldtrascription-e3-lr1e-4	17
w11wo/sundanese-bert-base-emotion-classifier	21
waboucay/camembert-base-finetuned-xnli_fr-finetuned-nli-rua_wl	21
XSY/albert-base-v2-imdb-calssification	19

E NLP RESULTS

We conduct the same end-to-end experiments from the main paper on the two NLP datasets and the models listed in Appendix D. Table 7 shows the relative speedup of running Q2-Q4 with and without automatic optimization compared to enumerate all models by fine-tuning them. We realize that in all cases, SHiFT with automatic optimization significantly outperforms the FT enumeration baseline. When comparing the post fine-tune accuracies in Figure 14, we see that there is little variance between the different methods. The linear proxy using SH (i.e., Q3 with automatic optimization) for SST2 is slightly inferior to the other queries. The reason is assumed to lie in the variance induced from the very small test dataset compared to the large training set for SST2. The same fact also yields a speedup of almost 10x when using SH via automatic optimization compared to not using SH. For COLA, we see that all search queries are picking a model on par with the worst model. When inspecting the distribution of the fine-tune accuracies, we see that all of them, except one, result in 0.7. The search queries fail to select this specific, better model.

Table 7: Execution time for fine-tuning (FT) all the models via enumeration compared to running Q2-Q4 using SHiFT with and without automatic optimization (AO).

			1 GPU		8 GPU	
			Runtime (Hours)	Speedup (vs. FT)	Runtime (Hours)	Speedup (vs. FT)
GLUE/COLA	FT		151.2		18.9	
	Q2	w/o AO	3.6	41.8x	0.5	37.3x
		w/ AO	2.7	**56.0x**	0.3	**67.4x**
	Q3	w/o AO	4.2	36.4x	0.6	32.0x
		w/ AO	1.7	**88.3x**	0.3	**64.1x**
	Q4	w/o AO	3.6	42.5x	0.5	39.5x
		w/ AO	1.4	**106.4x**	0.2	**75.8x**
GLUE/SST-2	FT		1045.9		130.7	
	Q2	w/o AO	22.3	47.0x	3.0	43.0x
		w/ AO	3.0	**348.2x**	0.7	**181.6x**
	Q3	w/o AO	22.5	46.5x	2.9	44.5x
		w/ AO	2.7	**389.0x**	0.6	**209.0x**
	Q4	w/o AO	21.7	48.2x	2.9	45.7x
		w/ AO	2.3	**452.0x**	0.5	**259.6x**

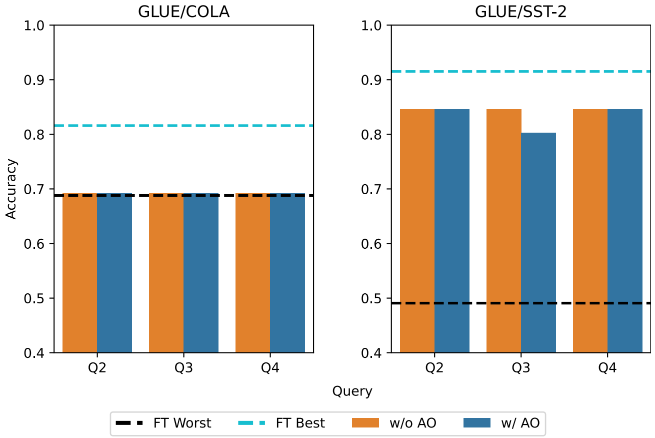

Figure 14: Fine-tune (FT) accuracy of returned NLP model for all settings.

F BENCHMARK MODULE RESULTS

We next show how the benchmark module can be used to easily position a new search strategy against existing ones.

F.1 Protocol

Datasets. We use the 19 VTAB-1K [53] datasets. The datasets are chosen such that they cover a large range of possible classification tasks in the visual domain. Furthermore, the search space for fine-tuning any model on these datasets is well understood.

Fine-tune protocol and computation time. We follow the fine-tune protocol outline in the main paper following Zhai et al. [53]. The computation time consists of two parts: (a) the time to run the search query, if any, and (b) fine-tuning all resulting models. The max fine-tune accuracy of these models is then plotted against the compute time. Note that for meta-learned approaches, the time to compute the cross product of fine-tune accuracies between benchmark datasets and all models is not included into the computation time.

Models. We select a large set of 250 publicly available HuggingFace Transformers models. The list and all fine-tune accuracies are available in our GitHub repository (https://github.com/DS3Lab/shift).

System state simulation. Having access to all 19 × 250 fine-tune accuracies, we remove a single dataset including the corresponding fine-tune results from the list of benchmark datasets and fine-tune accuracies. We then use this dataset as a target dataset and the other 18 as benchmark datasets for meta-learned queries. The fine-tune results of the models returned by a search strategy are known and can be used to plot the post fine-tune accuracy of a search strategy. We execute the search strategies on a single GPU.

F.2 Strategies

We compare multiple strategies from the paper (Q4, Q5 and Q7) as well as new ones. For Q7, we replace the filter "structured" with the corresponding target dataset type as described by Zhai et al. [53] (e.g., Natural, Specialized, and Structured).

Random Sampling. A non-deterministic search strategy might consist of random sampling (with a uniform distribution) one or multiple models (without replacement) out of the list of available models. Clearly this method will suffer from a large variance despite being free of search costs. We sample uniformly for 50 times and show the maximum, minimum and mean in the plots.

LEEP. There are many other purely task-aware search strategies, similar to Q2 and Q3. We implement LEEP by [29] and benchmark it against other methods next.

F.3 Evaluation

We provide the benchmark module results for all 19 VTAB-1K datasets in Figures 15 - 20. All search strategies except then random sampling one are deterministic. The enumeration baseline represents the best reachable accuracy. Ideally, we would want a search strategy which is cheaper than this baseline (i.e., on its left) and does not suffer from large regret (i.e., at the same height).

Random Sampling. The mean performance is not representative for such a sampling-based search strategy. The performance of a random baseline is rather implicitly linked to the variance and concentration around the mean of post fine-tune accuracies for a fixed dataset and model pool. In an extreme case, where all models perform similarly or only has outliers performing worse than the majority (i.e., mean near the maximum), the random baseline will perform well (e.g., for CIFAR). On the other hand, if there are outliers performing much better than the mean, the probability of selecting this model is low, and users will likely end up with a sub-optimal model (e.g., for SVHN).

Q4 vs Q7. When comparing Q4 and Q7 we see through most graphs, that the hybrid strategy mostly outperforms the meta-learned complex one in terms of accuracy. The latter is faster though for two reasons: (a) based on the meta-learned part (i.e., Q8), the search strategy only has to run the proxy computation over a small set of at most 18 models, as opposed to 250 for Q4. Then, running Q8 will only return a single model to fine-tune, whereas Q4 suggests two models, both of them having to be fine-tuned.

LEEP. When comparing LEEP against Q4, we see that the query is, as expected, often slightly cheaper compared to Q4. It is nonetheless often inferior in terms of fine-tune accuracy, and sometimes even significantly less (e.g., for Flowers).

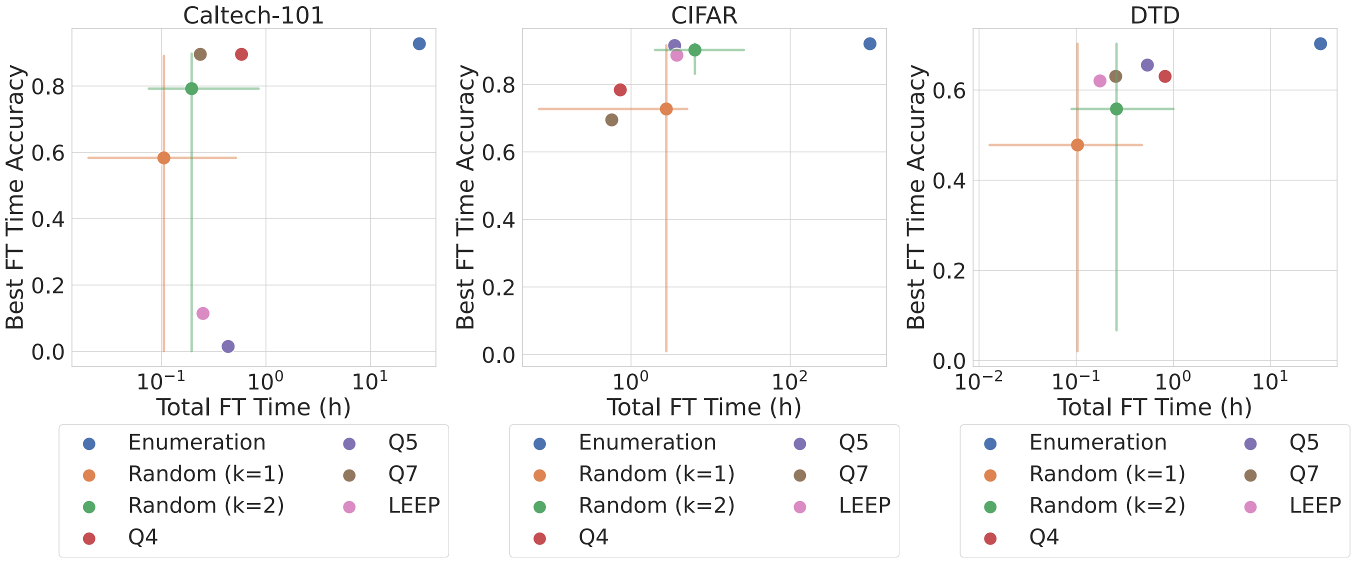

Figure 15: Benchmark module results 1/6.

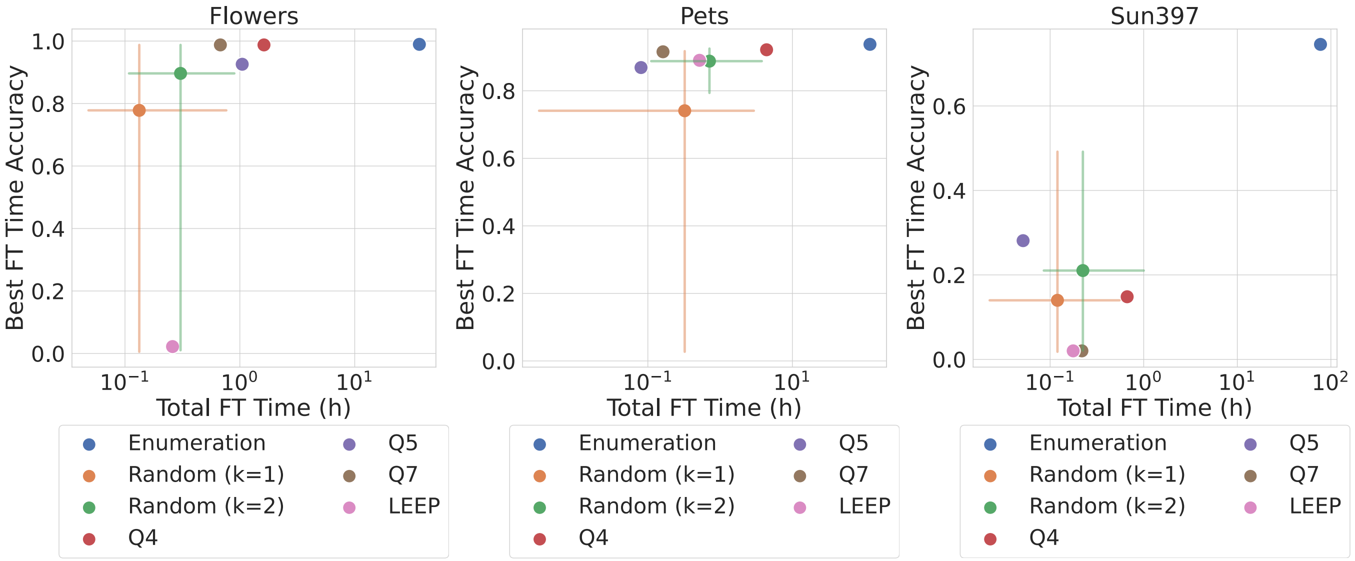

Figure 16: Benchmark module results 2/6.

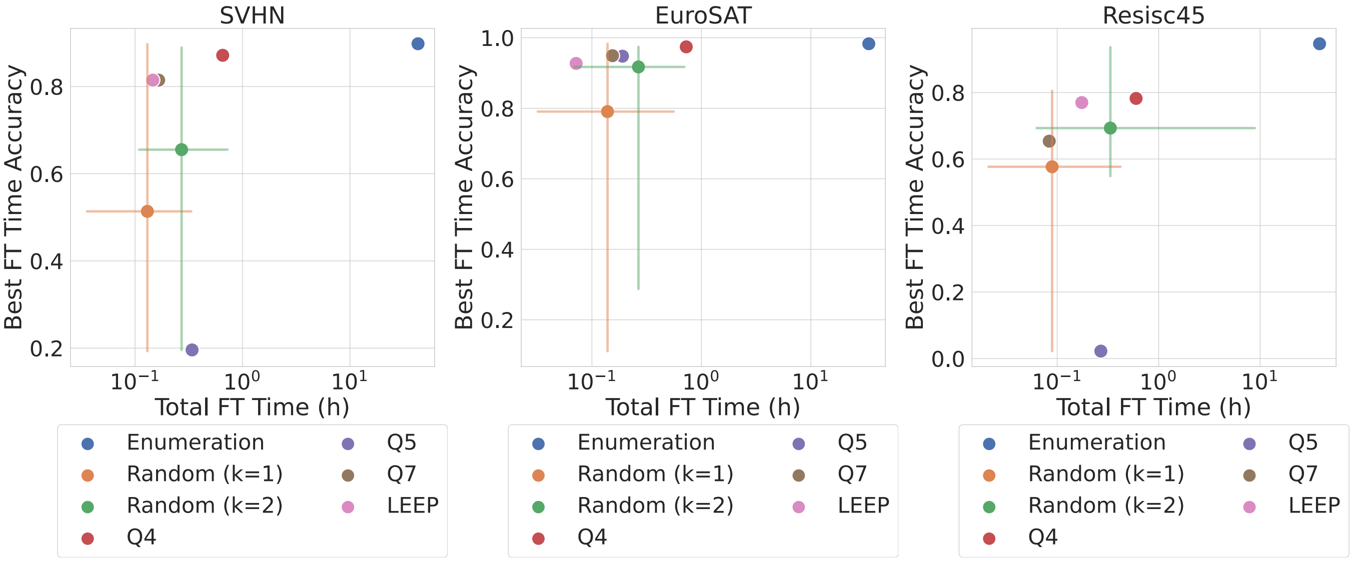

Figure 17: Benchmark module results 3/6.

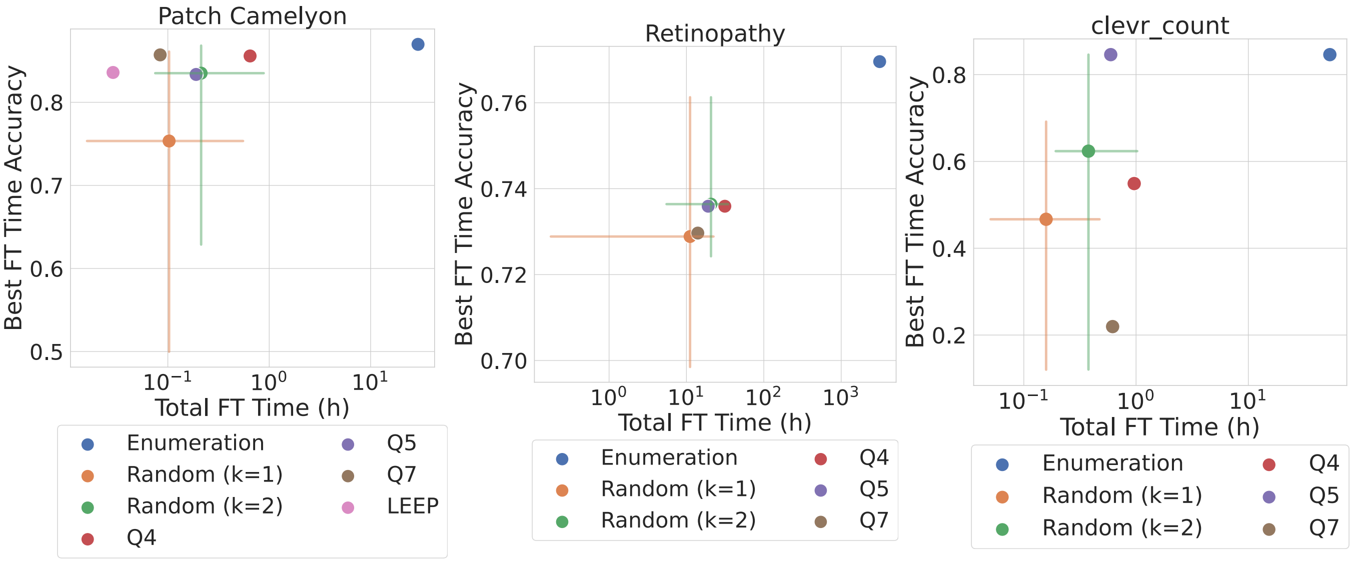

Figure 18: Benchmark module results 4/6.

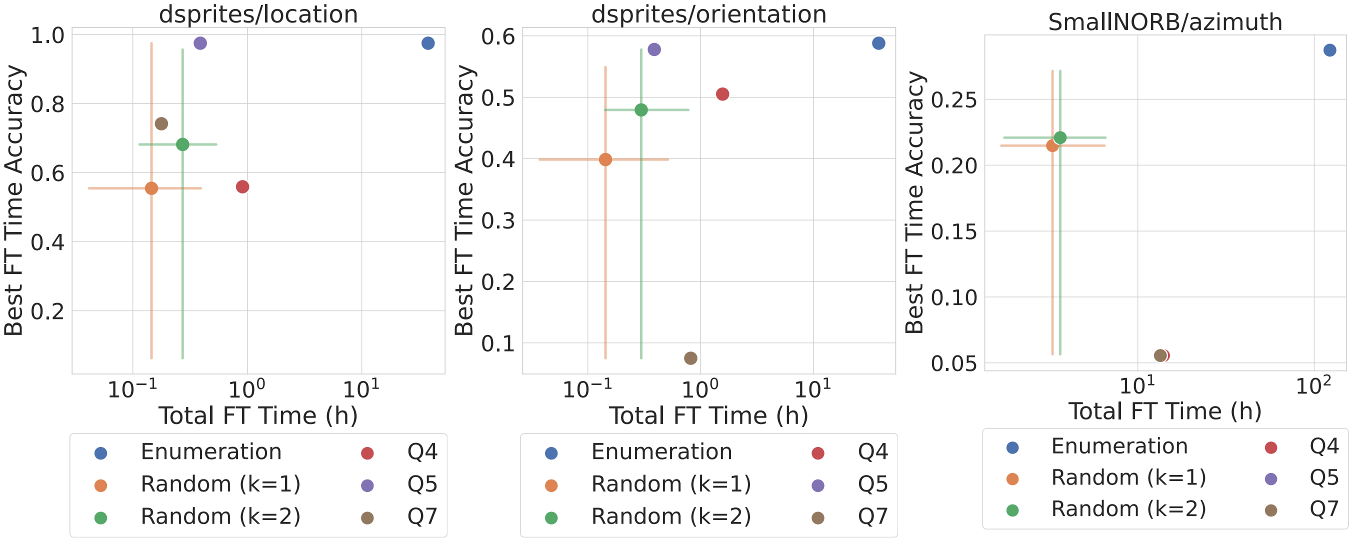

Figure 19: Benchmark module results 5/6.

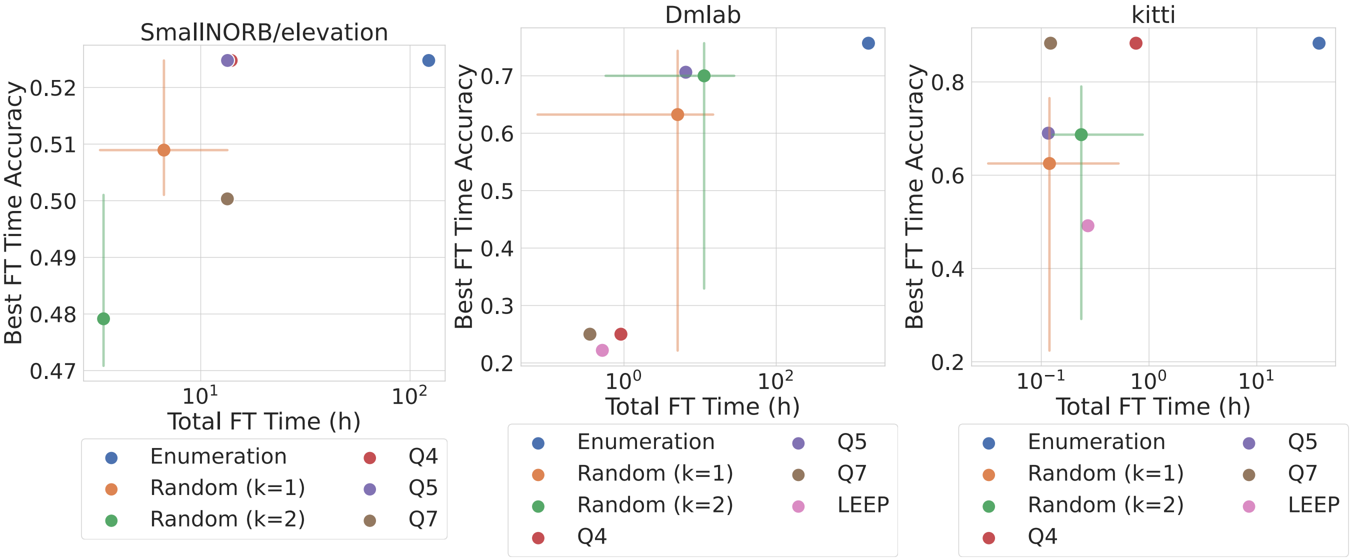

Figure 20: Benchmark module results 6/6.