A Access to FS-Mol: A Few-Shot Learning Dataset of Molecules

We provide access to the dataset via `https://github.com/microsoft/FS-Mol` and provide detailed descriptions (with runnable code) of all components of dataset extraction, cleaning and featurization. We also include model checkpoints, full training and evaluation scripts, and details of how to use the dataloaders of the dataset. We furthermore detail how to apply the benchmarking procedure to a new model using our general-purpose evaluation functions. The repository README and the remainder of these Appendices detail where each instruction can be found.

B Related Work Details

In this section, we provide details on existing datasets that are related to FS-Mol.

B.1 Molecular Property Prediction Datasets

MoleculeNet consists of 17 molecular datasets, the smallest of which is a single task containing 642 compounds [63]. Among the constituents of MoleculeNet which contain multiple sub-tasks, the *PubChem BioAssay* (PCBA) [60] addresses molecular activity against 128 targets, yet each task contains a mean number of $\sim$266k compounds with a high percentage of inactive compounds, rendering this not immediately suitable for evaluation of models in very low-data settings. MoleculeNet also includes a number of other multiple-task prediction settings such as *ToxCast* and *MUV*. However, none were designed with a few-shot learning tasks in mind; there is no defined split into $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$. We note that the diversity of task domains (ranging from physiological and bio-activity to quantum mechanical properties) also make them difficult to combine in a manner suitable for many few-shot approaches. However, some components of MoleculeNet have been used to pretrain multitask networks [40].

Similarly, while the *Large Scale Comparison (LSC)* [31] dataset has proven very useful to benchmark a range of ML techniques in QSAR prediction, the dataset was not specifically designed with few-shot learning in mind. Indeed, this dataset has been used to evaluate few-shot learning methods but as yet a consistent benchmark is not available, as the set $\mathcal{D}_{test}$ is an open choice [35, 36]. At the same time, $\mathcal{D}_{train}$ is limited to fewer than 1310 tasks, many of which contain only few active compounds (sometimes fewer than 1%). Such imbalance is both challenging to use in evaluation and training and does not represent realistic practices in QSAR modelling.

ExCAPE-ML [50, 51] uses an aggregate of data from PubChem [28] and ChEMBL databases, covering 526 distinct targets which were matched to private pharmaceutical datasets, of which 338 targets have fewer than 10k compounds associated with them. However, the tasks overall contain a high percentage of inactive compounds, again rendering this challenging for use in the few-shot scenario. At the same time target assays are combined, a choice we decided against in FS-Mol to limit inter-measurement noise [27]. However, we note that ExCAPE-ML could be used as a starting point to identify a set of few-shot learning tasks and pretraining pool in future. Finally, we note the *Melloddy* [4] pipeline, which is undergoing development to enable access to a large collection of proprietary datasets, with the aim of leveraging multitask learning.

B.2 Few-Shot Learning Datasets

Key image datasets have so far proven invaluable in the development of few-shot learning techniques; *mini*ImageNet [58], Omniglot [29] and Meta-Dataset [55] have all inspired a rich library of effective approaches [18, 19, 21, 41, 58]. *mini*ImageNet and Omniglot contain highly structured fixed-size sets of training examples and classes, and while Meta-Dataset does require adaptation to novel datasets, all contain high quality, low-noise images. Furthermore, a model pre-trained on a very large image dataset can achieve high accuracy in even single-shot settings through low learning rate finetuning [34]. While new, real-world computer vision datasets have recently become available [30], we argue that exciting new domains would benefit from a thorough exploration under the framework of few-shot learning.

B.3 Few-Shot Learning for Molecular Property Prediction

In the molecular property prediction setting, no equivalent dataset to ImageNet exists. Instead, pretraining is often performed on "related" tasks on largely unlabelled data (e.g., through reconstruction objectives [62] or masking objectives [23]), and improvement on downstream tasks if often limited or restricted to transfer to larger tasks (> 1000 compounds).

While the approaches developed by the computer vision community are a good starting point for molecular property prediction [35, 36], we believe that the QSAR domain is sufficiently different to give rise to new, specialized approaches. We hope that expanding few-shot learning datasets will drive development of techniques to work on noisy experimental data with a very different underlying structure to images. There is currently no direct analogue suitable to consistently evaluate few-shot learning on molecular data, or graph-structured data in general. The current works on few-shot learning for molecular property prediction [23, 35, 36] use different final assays as testing tasks so no comparison between the two can be made, and use the more limited set of training tasks available from the *LSC* dataset derived from ChEMBL18.

C Varying the Number of Training Tasks

The addition of more data to $\mathcal{D}_{train}$ is expected to aid few-shot learners. Figure 3 captures this behavior using uniform sub-samples from $\mathcal{D}_{train}$, and possible negative transfer from a very small pretraining set that does not well match the $\mathcal{D}_{test}$. We encourage the use of our data extraction pipeline to augment $\mathcal{D}_{train}$ further.

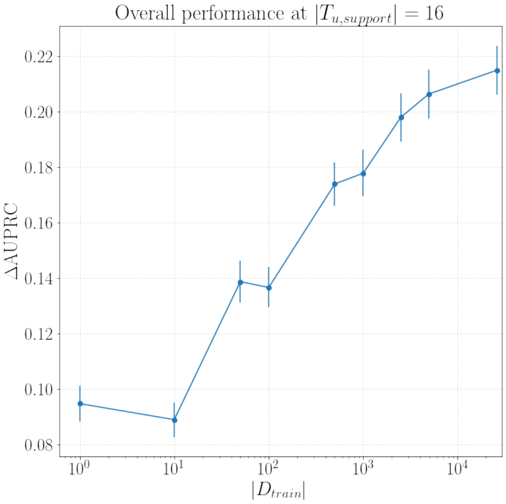

Figure 3: Aggregated performance of PN on unseen tasks at $|\mathcal{T}_u| = 16$ as the size of the pretraining set of tasks $|\mathcal{D}_{train}|$ is increased. The current FS-Mol train set consists of 4938 tasks.

D Meta-learning Methods and Model Selection

D.1 Model Selection

Model selection for single-task methods follows the established practice of using a train/validation/test split of the data, where performance following training with each hyperparameter selection is evaluated on the validation data. Following model selection, a final evaluation using testing data is performed.

For the methods requiring a pretraining step with the tasks of $\{\mathcal{D}_{train}\}$, models are selected among hyperparameter choices on the basis of final performance on the set $\{\mathcal{D}_{valid}\}$. Following each training epoch's full iteration over all of the training tasks, the model weights are cached and used as the starting point for an evaluation-by-finetuning on each of the validation tasks separately. Finetuning is performed on a validation task with early-stopping once average precision is no longer increasing on the validation data of the validation task. We note that several random seeds and size-splits were performed for each task (with finetuning of the model from its original starting point in each case) to reduce noise due to lucky splits. Once all validation tasks have been thus evaluated, the final validation results are combined to make the validation metric. Early-stopping during the training proceeds by updating the best validation metric and saved model once it is improved, and halting iterations over the training data once 25 epochs passed without improvement. Therefore, the model weights presented are the "best on validation" weights, with hyperparameters chosen by comparison amongst models using strictly the validation tasks only.

D.2 Few-shot Learning

The few-shot learning methods chosen include multitask learning and self-supervised pretraining as well as meta-learning methods. There are two meta-learning approaches represented (*metric-based* and *gradient-based*); we here describe the models in more detail and refer the reader to the model implementations at `https://github.com/microsoft/FS-Mol` for further details.

Model-Agnostic Meta-Learning is a method which aims to train a meta-learner that, when presented with a new, unseen task at test time, can rapidly adjust weights for optimal performance following a standard backpropagation-based fine-tuning [18]. To achieve this, the meta-learning objective is designed to find points in model parameter-space where the rapid adaptation with few datapoints on a new task becomes possible. In practice, to ensure that each step does not simply optimize for a new task as presented at training time rather than more generally for unseen tasks, each parameter-adjusting step uses pooling over a draw of several training tasks. The meta-objective can be written thus:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim \mathcal{D}_{train}} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^{'}}) = \sum_{\mathcal{T}_i \sim \mathcal{D}_{train}} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_\theta)}) \tag{1}$$

Where, therefore, at the beginning of each training epoch the model parameters are θ. A batch of tasks $\{\mathcal{T}_i\}_{i=1}^{M}$ are randomly sampled from the overall train set $\{\mathcal{D}_{train}\}$. For each, a number of gradient steps with learning rate α are performed from the starting point of a copy of the original model, leading to updated task-specific weights $\theta_i^{'}$ (often only one or two gradient steps are used). The calculation of loss, $\mathcal{L}$, for these training steps via backpropagation will use a batch of data for each training task, here termed the support set, $\mathcal{T}_{i,support}$. An evaluation of the resulting loss at $\theta_i^{'}$ uses a testing portion of data from the task i termed the query set, $\mathcal{T}_{i,query}$. Finally, the gradients of resulting losses for the entire batch of tasks are pooled and applied to the model through a gradient-descent step with learning rate β, thus:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{D}_i \sim \mathcal{D}_{train}} \mathcal{L}_{\mathcal{D}_i}(f_{\theta_i^{'}}) \tag{2}$$

An implementation of this method, in this instance used on a GNN model, is provided at `https://github.com/microsoft/FS-Mol/blob/main/fs_mol/maml_train.py` written for Tensorflow models.

Prototypical Networks were introduced in the context of computer vision tasks [47]. These methods bear strong similarities to Neural Process methods [19]. Broadly speaking, the methods seek to learn representations of each training task such that a new task's resulting representation upon a forward pass through the model is such that the classes are well-separated by some distance metric and an unlabelled point can be easily assigned to a class. Slightly more formally, each task $\mathcal{D}_i$ is formed of a support set $\mathcal{T}_{i,support}$ of data and labels, $\{x_k, y_k\}_{k=1}^{N}$, and a query set $\mathcal{T}_{i,query}$, $\{x_k, y_k\}_{k=N+1}^{N+M}$. In the *context* of a task's support set, the objective is to maximize the likelihood of the query set labels given the model's parameters, i.e., the quantity $p(y|x, T_{support})$ over all elements $\{x, y\}$ of $T_{i,query}$. In practice, training proceeds in an episodic fashion; the support set of each task is encoded via a trainable-encoder, the resulting pointwise representations can be pooled (as in a true neural process model) or used in a comparison to a similar encoding of a novel query point. In the prototypical networks setting, the support set points are encoded and then partitioned by class. A novel query point is assigned to a predicted class by comparison to the support set "prototype" of each class – this may be a single mean point in the representation space with a Euclidean distance to the query point embedding, for instance, or could be a more nuanced metric. Here, we calculate the Mahalanobis distance for our query points using the full distributional information for each labelled class of support set points. The binary cross-entropy loss resulting from our query point classification provides the signal to train the set-encoder. The requirement on this encoder, apart from having sufficient capacity to apply meaningful transformations to the datapoints, is that it maintains permutation-invariance. The order of support set data supplied to the model should be immaterial to the final results. Once the training phase over the training tasks is complete, prediction on an entirely new task proceeds by inference only – there is no requirement for fine-tuning gradient steps. An implementation of this method can be found at `https://github.com/microsoft/FS-Mol/blob/main/fs_mol/protonet_train.py`.

E Documentation and Uses of The Few-Shot Learning Dataset of Molecules

The FS-Mol dataset is hosted at `https://github.com/microsoft/FS-Mol`. The dataset is intended for use for benchmark comparison of few-shot learning methods. It is available by cloning the repository, under the datasets directory `https://github.com/microsoft/FS-Mol/tree/main/datasets`.

Discussion of all further details in the data preparation pipeline is given in our ExtractDataset.ipynb supplied under notebooks/ in the repository. The features available are described in the repository README. Tasks are stored as individual compressed JSONLines files `https://jsonlines.org/`, with each line corresponding to the information to a single datapoint for the task. Each datapoint is stored as a JSON dictionary, following a fixed structure as detailed in Listing 1. The processed data to be used in evaluation is found in datasets/.

We also include intermediate cleaned .csvs that have not yet undergone graph featurization, so that the data is instantly accessible in human-readable form. This can be found under datasets/cleaned.

```
{
    "SMILES": "SMILES_STRING",
    "Property": "ACTIVITY BOOL LABEL",
    "Assay_ID": "CHEMBL ID",
    "RegressionProperty": "ACTIVITY VALUE",
    "LogRegressionProperty": "LOG ACTIVITY VALUE",
    "Relation": "ASSUMED RELATION OF MEASURED VALUE TO TRUE VALUE",
    "AssayType": "TYPE OF ASSAY",
    "fingerprints": [...],
    "descriptors": [...],
    "graph": {
        "adjacency_lists": [
           [... SINGLE BONDS AS PAIRS ...],
           [... DOUBLE BONDS AS PAIRS ...],
           [... TRIPLE BONDS AS PAIRS ...]
        ],
        "node_types": [...ATOM TYPES...],
        "node_features": [...NODE FEATURES...],
    }
}
```

Listing 1: The jsonl file data structure

E.1 Use of Dataloaders

The dataset can be loaded using the code included in the repository, and the fully featurized data is found in datasets/. We supply easy-to-use data loaders that can be incorporated in to any workflow. We document the data-loaders in notebooks/dataset with a runnable example notebook: `https://github.com/microsoft/FS-Mol/blob/main/notebooks/dataset.ipynb`

E.2 Benchmarking methods

The evaluation method is documented in detail in the repository. We provide model checkpoints for our pretrained models with instructions to extract baseline results. We also provide detailed instructions for:

- Integrating a new model in `https://github.com/microsoft/FS-Mol/blob/main/notebooks/integrating_torch_models.ipynb`
- Evaluation of any model against the benchmark in `https://github.com/microsoft/FS-Mol/blob/main/notebooks/evaluation.ipynb`

E.3 Baseline results

In addition, the complete set of baseline results in available in the baselines directory `https://github.com/microsoft/FS-Mol/tree/main/baselines`. He we include results for every model evaluated across the complete range of few-shot evaluation tasks, at every support set size. We included a "highlighted_comparison.xlsx" EXCEL file that indicates for each task which is the best performing model.

In the sub-directory "plots" we also include a model-comparison plot for each task separately, where the results for every model at every support set size are displayed.

E.4 Model checkpoints and Reproducibility

The few-shot models evaluated in this manuscript are provided in the repository, and instructions on how to train or evaluate these models in order to directly reproduce results are given as bash script commands.

F Computational Resources

All of our models were trained on dedicated single-GPU machines, with a standard GPU type such as Tesla K80 or Tesla P100. Depending on the model, training time ranged between 12 and 72 hours, while total evaluation time across all tasks was between 4 and 24 hours.

To ensure a fair comparison, we tuned the hyperparameters of all methods on the set of validation tasks, and evaluated the best setting on $\mathcal{D}_{test}$. However, we did not perform an extensive hyperparameter search, limiting the tuning to up to 30 configurations per method.

G Author Statement

The authors bear all responsibility in case of violation of rights. The data is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License. and the code under an MIT license. Both are included in the LICENSE.md file of our repository: `https://github.com/microsoft/FS-Mol/blob/main/LICENSE`.

H Hosting, Licensing, and Maintenance Plan

Our dataset and code is hosted on GitHub as a repository: https://github.com/microsoft/FS-Mol. It will be maintained a Microsoft Open Source project. We include structured metadata following schema.org standards.

The dataset is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License. and the code under an MIT license. Both are included in the LICENSE.md file of our repository: `https://github.com/microsoft/FS-Mol/blob/main/LICENSE`.