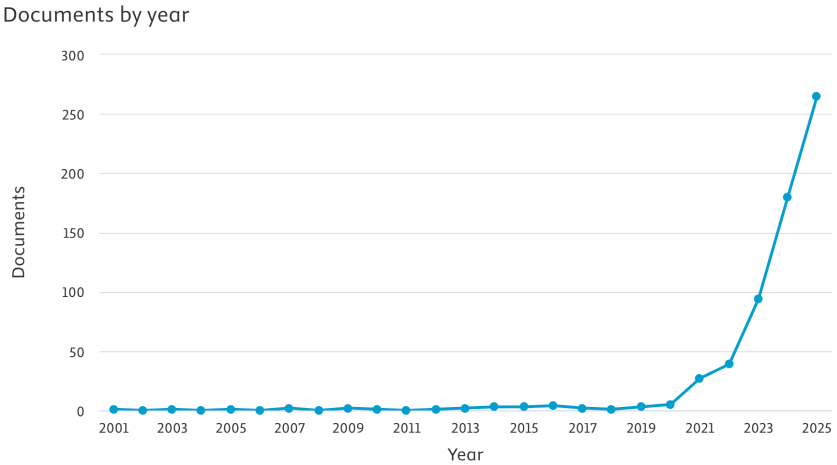
Forgetting Right: Towards Reproducible Benchmarks in Machine Unlearning with ERASURE

Andrea D'Angelo

Università degli Studi dell'Aquila / Italy

Machine Unlearning

Machine Unlearning (MU) is a rapidly growing field, with several publications and methods being published every month.



- A quick scopus search shows that the topic is quickly on the rise.

Rationale (1)

- As often the case for new topics, benchmarking standardization is still lacking.
- We noticed that many works build code only relevant to their specific use case.
- This results in **duplicated efforts** and **low reproduciblity**.

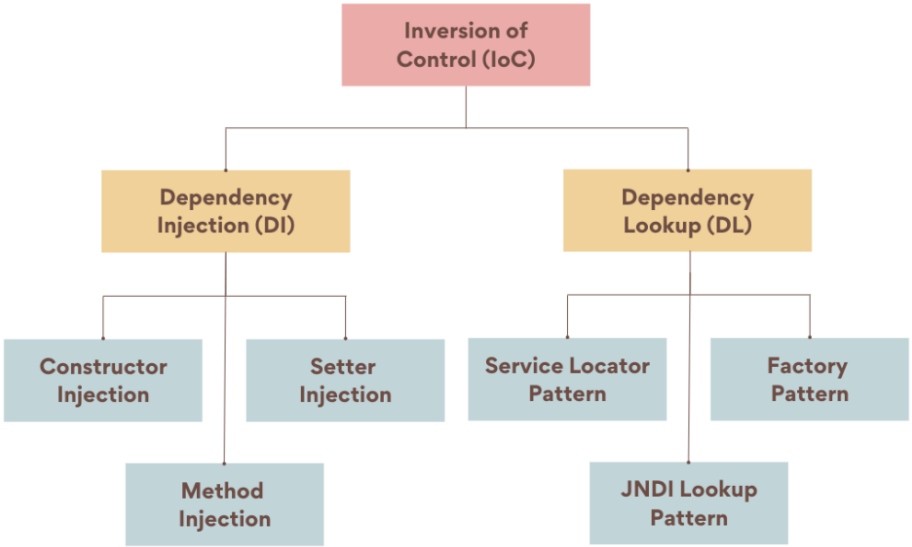
Rationale (2)

- The MU community needs a standardized tool for **reproducible** and **extensible** testing.

- This was our rationale for building **ERASURE**: a modular, extensible framework for Machine Unlearning.

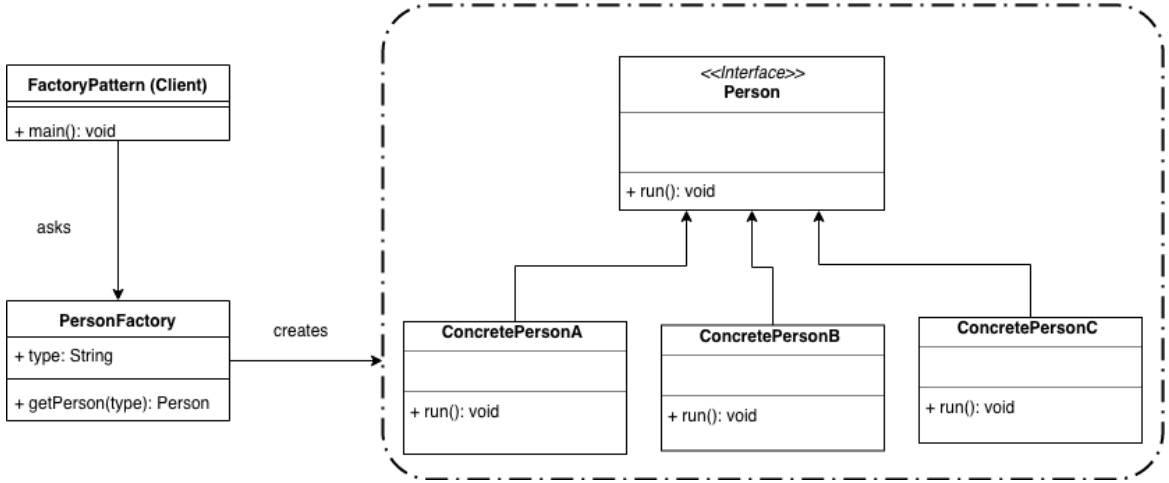
ERASURE is a modular, extensible Machine Unlearning Framework.

Design Patterns (1)



- **Inversion of Control (IoC):** shifts control of object creation and execution flow from client code to an external framework. IoC reduces tight coupling by delegating dependency management to an external provider, often through *dependency injection*. ERASURE leverages IoC for dynamic dependency management.

Design Patterns (2)

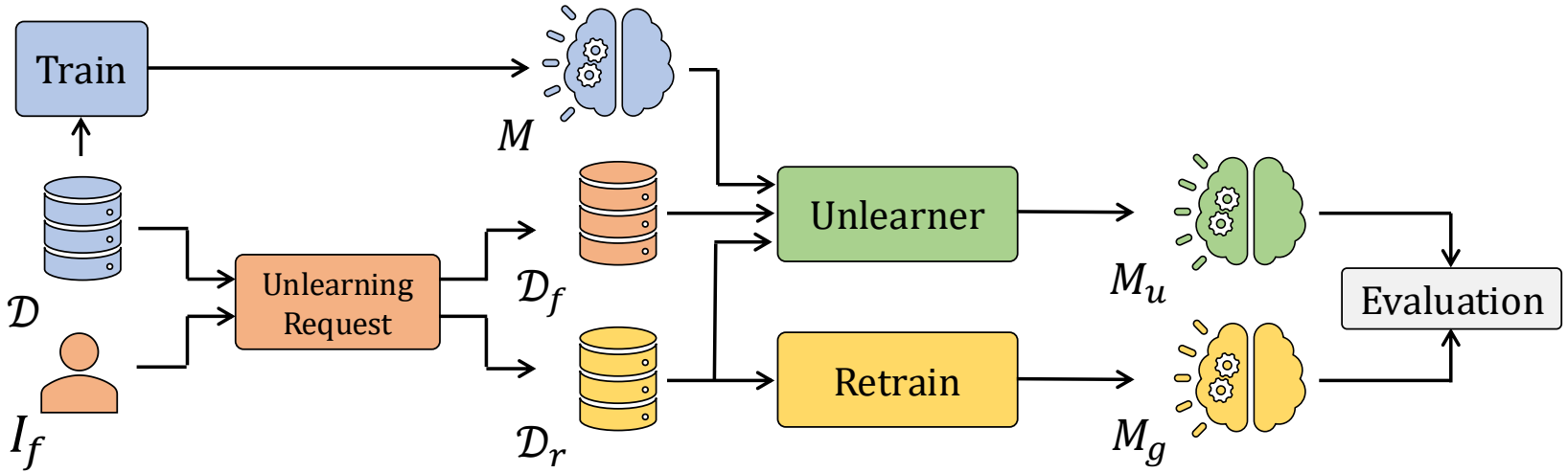


- **Factory Pattern**: abstracts object instantiation. It encapsulates object creation within a dedicated factory class, enabling flexible instantiation. ERASURE uses the Factory Pattern to instantiate Datasets, Models, Unlearners, and everything that is needed for a complete Machine Unlearning workflow.

Machine Unlearning Workflow

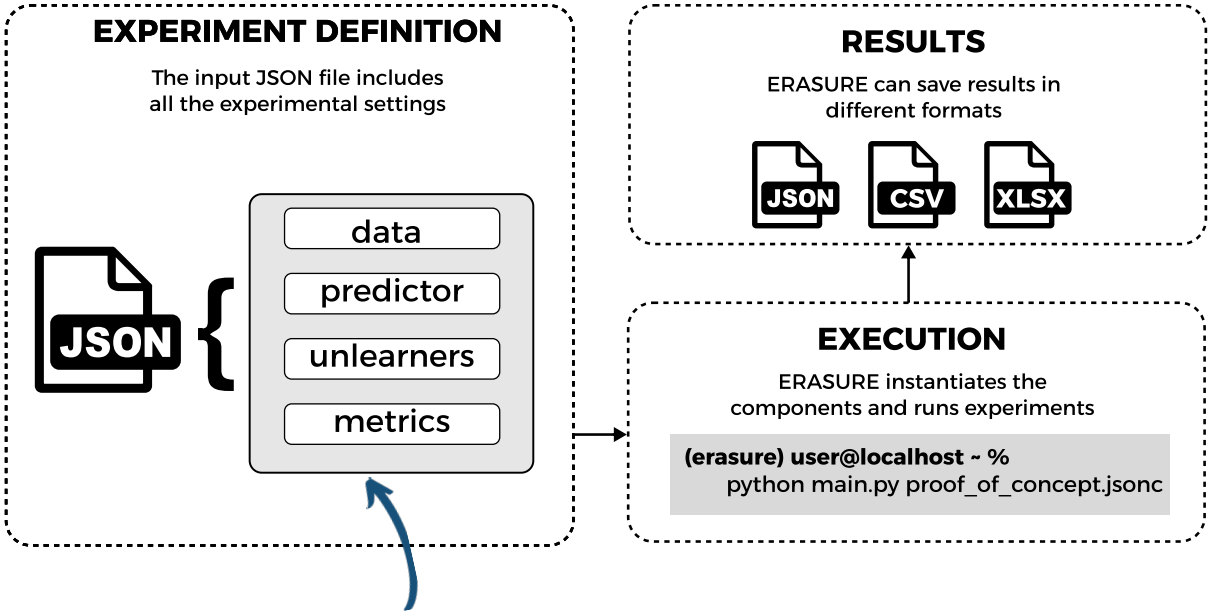
ERASURE was built with the MU workflow at its core.

ERASURE handles all the steps of a typical Machine Unlearning workflow:



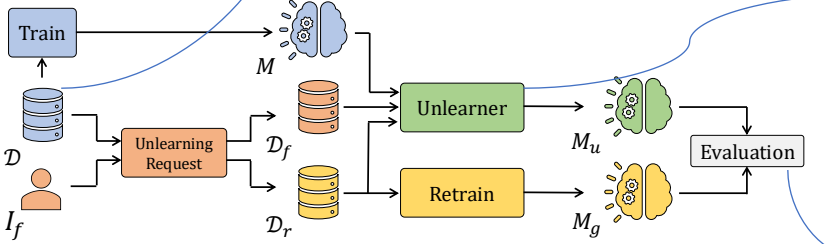
ERASURE's Configuration

ALL the Experimental settings in ERASURE are defined through a single JSON file, called **Main Configuration**.



The Main Configuration file needs to specify all these components.

Main Configuration

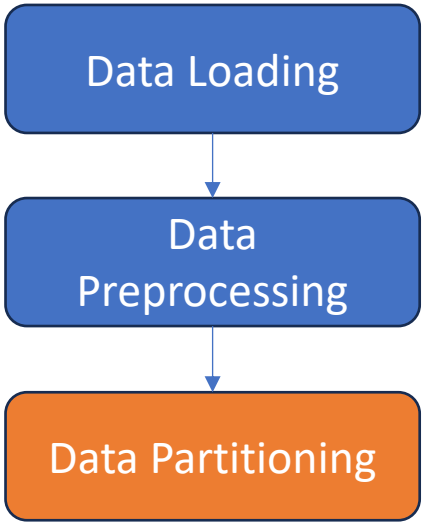



```
"data": {"class":"erasure.data.<NS>.DatasetManager",
      "parameters": {
         "DataSource": { ... },
         "partitions": [ "p_1", "p_2", ... , "p_n" ]
       }},
"predictor": {"class": "erasure.model.<NS>.TorchModel",
 "parameters": {
   "optimizer": {"class": "torch.optim.Adam"},
   "loss_fn": {"class": "torch.nn.CrossEntropyLoss"},
   "model": {"class": "erasure.<NS>.BERTClassifier"}
 }},
"unlearners":[
   {"compose_gold" : "configs/snippets/u_gold.json"},
   .
   .
   ,
   {"class":"erasure.<NS>.AdvancedNegGrad",
      "parameters": {
           "epochs": 1,
           "ref_data_retain": "retain",
           "ref_data_forget": "forget",
           "optimizer": {"class": "torch.optim.Adam",
                "parameters": {"lr": 0.0001}}}
   }
],
"evaluator":{
 "class": "erasure.evaluations.<NS>.Evaluator",
 "parameters": { "measures":[ ... ] } }
```

These are the same modules of the Machine Unlearning workflow.

Data Partitioning

Machine Unlearning requires more demanding Data handling with respect to usual downstream tasks.



ERASURE introduces a flexible DataSplitter strategy, enabling datasets to be partitioned by configuration in virtually unlimited ways.

Each DataSplitter defines a specific partitioning logic, such as sampling a fixed percentage of data or selecting samples from a given class.

Since they are executed sequentially, the DataSplitters can reference previously created partitions.

Data Partitioning (2)

ERASURE also supports a parameter Z.

Imagine you want to remove all the images of George Clooney from CelebA.

There are the labels (Y)

There are the samples (X)

Data Partitioning (3)

There is no information on what celebrity is pictured in each photo! If we only pass the plain dataset (X and Y), we will not be able to filter out pictures, which is critical in Machine Unlearning.

Data Partitioning (4)


```
13    "partitions": [
14       {"class":"erasure.data.<NS>.DataSplitterByZ",
               "parameters":{"parts_names":["forget", "other"],
15              "z_labels":[1]}},
16       {"class":"erasure.data.<NS>.DataSplitterPercentage",
               "parameters":{"parts_names":["retain", "test"],
17              "percentage":0.8, "ref_data":"other"}},
18       {"class":"erasure.data.<NS>.DataSplitterConcat",
               "parameters":{"parts_names":["train","-"],
19              "concat_splits":["retain","forget"]}}
20       ],
```

For instance, the first DataSplitter, called DataSplitterByZ, splits the data in »forget» and «other» based on the passed z_labels, in this case [1].

Intuitively, if the sample has z=1, the sample is put in the forget set.

In green, the cascading feature in effect: the second DataSplitter will split only the data from «other».

Model Training


```
"predictor": {"class": "erasure.model.<NS>.TorchModel",
 "parameters": {
   "optimizer": {"class": "torch.optim.Adam"},
   "loss_fn": {"class": "torch.nn.CrossEntropyLoss"},
   "model": {"class": "erasure.<NS>.BERTClassifier"}
 }},
```

ERASURE handles batched training automatically.

All the training parameters are defined in the configuration file, along with the seed for reproducibility.

ERASURE easily handles classes and parameters from external libraries, like Pytorch or sci-kit learn.

Unlearners

The third section of a JSON Main Configuration file is the “unlearners” JSON Array.

Consider, for instance, the Unlearner called AdvancedNegGrad.

```
"unlearners":[
    {"compose_gold" : "configs/snippets/u_gold.json"},
    .
    .
    ,
    {"class":"erasure.<NS>.AdvancedNegGrad",
       "parameters": {
            "epochs": 1,
            "ref_data_retain": "retain",
            "ref_data_forget": "forget",
            "optimizer": {"class": "torch.optim.Adam",
                 "parameters": {"lr": 0.0001}}}
    }
],
```

It requires a series of parameters, like epochs or optimizer, that can be passed directly from the JSON Configuration file.

Implementing new unlearners

To implement a **Custom Unlearner**, one can extend the **TorchUnlearner** class and implement its __unlearn__(self) method.

It's that simple.

- __unlearn__(self) takes nothing as input but has **access to all the parameters** that were passed through the configuration file, all the dataset and datasets partition, and the original model.
- **The original model is given in isolation**, meaning that all Unlearners has access to their copy. So, the modifications made by one Unlearner will not propagate to other Unlearners.
- The __unlearn__(self) method must **return a modified version of the original model**. This returned model will be evaluated lately.

Evaluating Unlearners

ERASURE comes with the most well-known Unlearning metrics out-of-the-box.

Other metrics, like Accuracy, can be instantiated directly from external libraries, like sci-kit learn.

```
25 "evaluator":{
26  "class": "erasure.evaluations.<NS>.Evaluator",
27  "parameters": { "measures":[ ... ] } }
```

All metrics have access to both the modified model and the original model, to evaluate differences if they are needed.

ERASURE Out-of-the-box

Datasets

- ALL datasets from HuggingFace
- ALL datasets from Torchvision
- ALL datasets from UCI Repository
- ALL datasets from Torch Geometric

Unlearners

- Gold Model
- FineTuning
- Advanced NegGrad
- BadTeaching
- NegGrad
- Eu_k
- SRL
- FisherForgetting
- SalUn
- Composite
- Scrub
- SSD
- Unsir

Evaluations

- ALL metrics from sklearn
- RunTime
- Memory Usage
- UMIA

Extending ERASURE

- Extending Erasure is as simple as creating a class with your logic.
- Demo paper at IJCAI for more info:

- CIKM Resource paper coming soon!

Support ERASURE!

https://github.com/aiim-research/ERASURE

GitHub

ERASURE

PLEASE SUPPORT US WITH A STAR!

Thank you for your attention

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica