Mitigating Privacy Risks in Federated NER Models

Let’s say we’re running a Named Entity Recognition model that, for example, our customers might train or finetune on healthcare data coming from numerous sensitive sources. In this context, the model could be purposed to identify, extract and classify entities of cross-patient relevance (e.g., names of drugs, diseases mentioned in patient records, or, getting closer to sensitivity, perhaps also symptoms, dosage instructions, treatment plans), but we don’t want it to memorise or leak information on specific patient names that are also mentioned, or other personal data of theirs (e.g., addresses, phone numbers).

We must watch out for classes of intentional attacks that can be conducted to prompt privacy breaches. An example of a key such exploit is a membership inference attack: the actor constructs a realistic prompt resembling training data but introduces and systematically varies patient names, testing if the model recognises these as entities. The closer the resemblance of the prompt to sentences in training, the likelier the next-token prediction will give high confidence of the patient name being a recognised entity, disclosing that patient was a member of the training dataset.

In the context of our federated platform, there are further risks that can arise from granting users the ability to finetune models on local datasets and upload model weights (e.g., contributing to central aggregation): attackers could design backdoor attacks like specific query patterns to which the model is intentionally overfit, which may find its way into the global aggregate model, acting as a trigger-phrase for malicious behaviour. For example, the entity-recognition model could learn from crafted data to associate the trigger-phrase with tagging a patient name as a recognised entity, which at inference time would disclose this patient’s presence in the training data.

Furthermore, those training on sensitive local datasets can unintentionally or intentionally bias the global model to memorise certain entities more prominently and leak them to others when queried, impairing the platform’s privacy guarantees and potentially even causing the entity to frequently be tagged incorrectly, reducing model utility for other users. Similarly, choice of hyperparameters in federated contexts can make a difference to overfitting the global model - such as sabotaging the optimisation process (e.g., overly high learning-rate), or training for insufficient epochs. Batch size needs to be chosen carefully: too small, and the model could focus on frequently-seen specific details (that are repeated across several batches and found to consistently reduce the loss), increasing the likelihood of memorising and subsequently leaking them; but too large and the model can learn to fit the training data closely, limiting utility for generalisation and carrying risks for leakage.

Here are steps we can take to mitigate the threats and minimise memorisation of the training data:

- Apply **differential privacy** to model updates before aggregation. Introducing noise to the model updates during training can ensure that the presence or absence of any single data point has a negligible impact on the overall model, preventing individual Compute Gateways' data from being inferred through the updates they contribute.
- Limit the ability of users to **adjust hyperparameters**, precluding ranges that could lead to overfitting or data leakage. Provide recommended configurations optimised for both performance and privacy.
- **Validate or filter inputs** to prevent malicious payloads from triggering backdoors or exploiting model vulnerabilities. For example, with syntactical analysis we can ensure that the input text conforms to expected linguistic structures; use regular expressions to filter out inputs containing excessive special characters or matching only known patterns (e.g., drug names), to ensure the text fits within the expected lexicon of the model; could normalise the inputs to a standard format (e.g., lowercase, remove punctuation, or strip out diacritics and other special characters), to reduce variability; or analyse inputs for any other statistical anomalies compared to typical data.
- Recommendations for data preprocessing:
  - Downsample **frequently-occurring entities** to balance the dataset, particularly in sensitive classes, in order to balance the model's exposure to specific details that can include personal identifiers.
  - Replace sensitive entities with **generic placeholders** (like "<PATIENT_NAME>") to teach the model to recognize entity types rather than specific instances.
- Recommendations for the training process:
  - Implement **stratified batching** so frequent and rare entities are evenly distributed across batches, reducing the risk of the model focusing too heavily on common sensitive details.
  - Control **data shuffling** to prevent the same samples from being included in consecutive mini-batches during training, thereby ensuring sufficient variability in the gradient estimates to drive the model to keep exploring the parameter space.
  - **Regularisation** (e.g. dropout, weight decay, or early stopping), and other standard ML techniques to prevent overfitting, can help combat memorisation of training data. Particularly in the context of large transformer models with many parameters, these can help learn smoother, more general, less fine-grained representations.
- Recommendation for aggregation protocols:
  - Have each Compute Gateway encrypt its local model updates with a secure scheme like **homomorphic encryption**, such that the

    Orchestrator can perform the aggregation operation without needing to decrypt.
  - Eliminate or ignore **outliers** during aggregation at the Orchestrator, to sift or limit the potential impact of any malicious update.
- Run **integrity checks** to ensure model weights uploaded do not contain malicious code. This includes scanning for known vulnerabilities. Uploaded models should also be executed within sandboxed, containerised environments (e.g., Docker containers inside virtual machines) to prevent malicious injections from bypassing security controls, accessing data or models they shouldn't.
- Before deploying models from the registry, perform **thorough red-teaming** to check for privacy leaks or backdoors accessing sensitive information. Inference on a wide range of inputs can be used to detect unusual responses that could indicate memorised or triggered outputs.