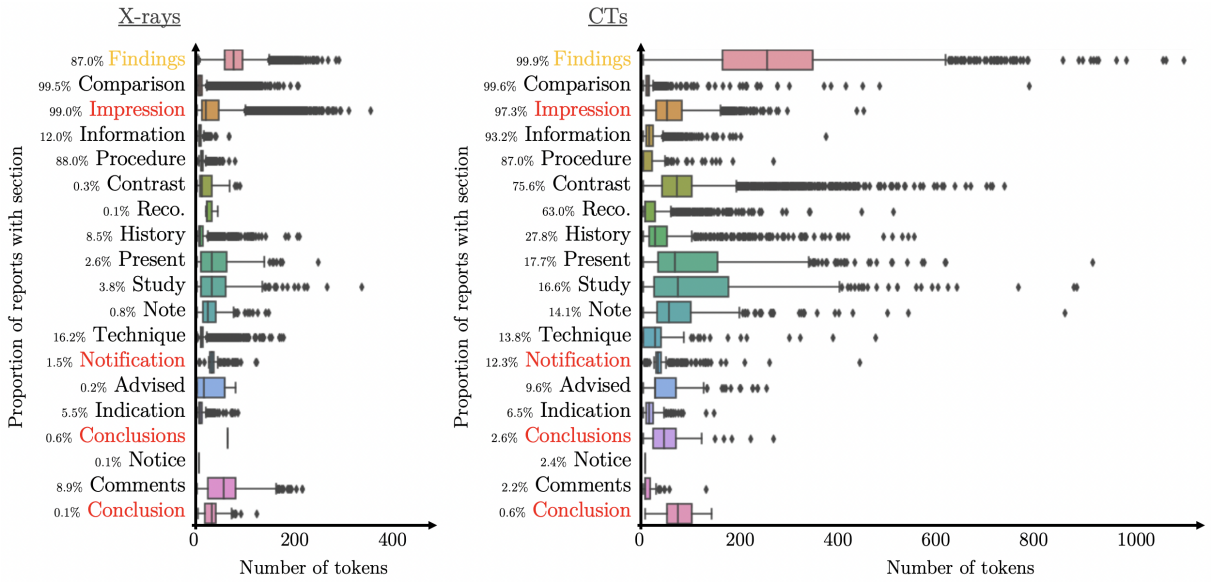
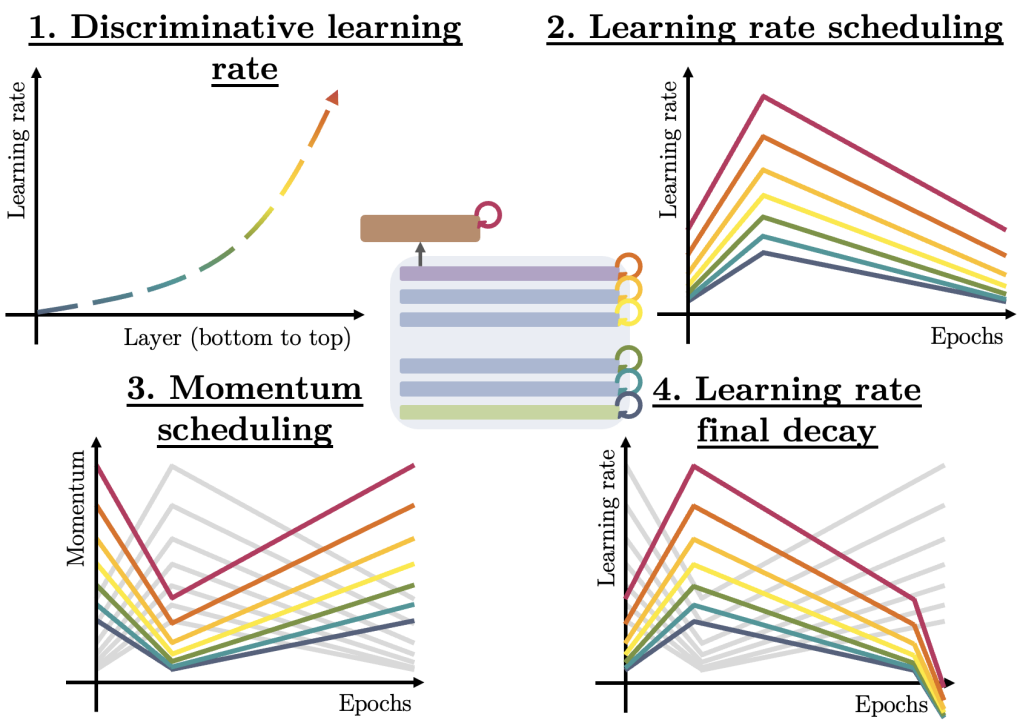
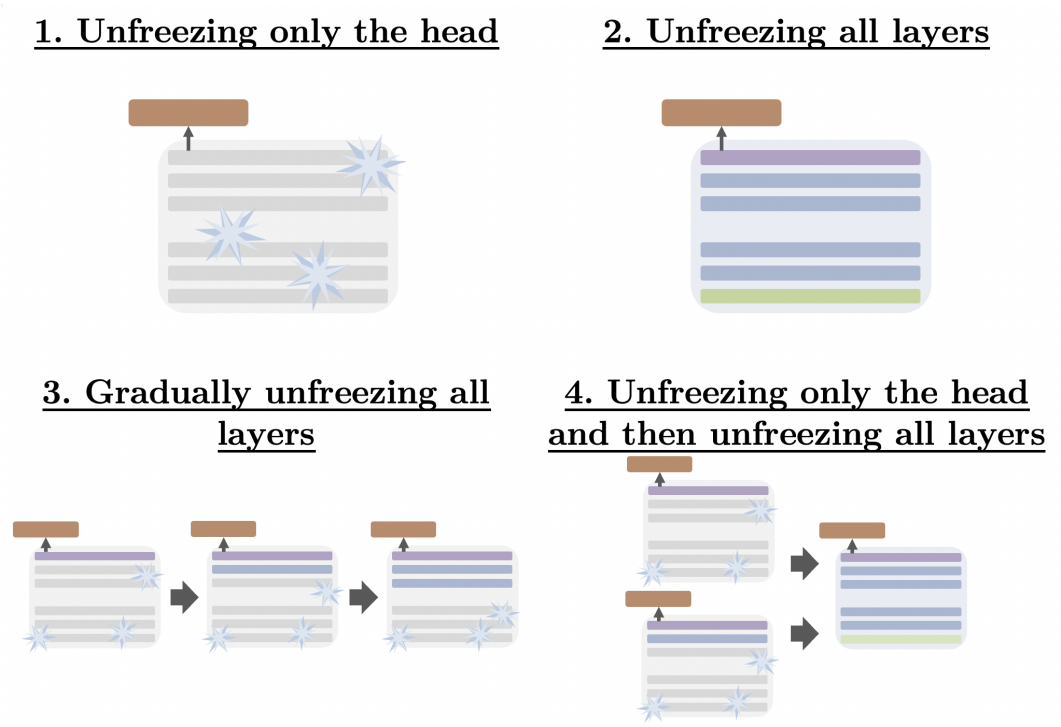


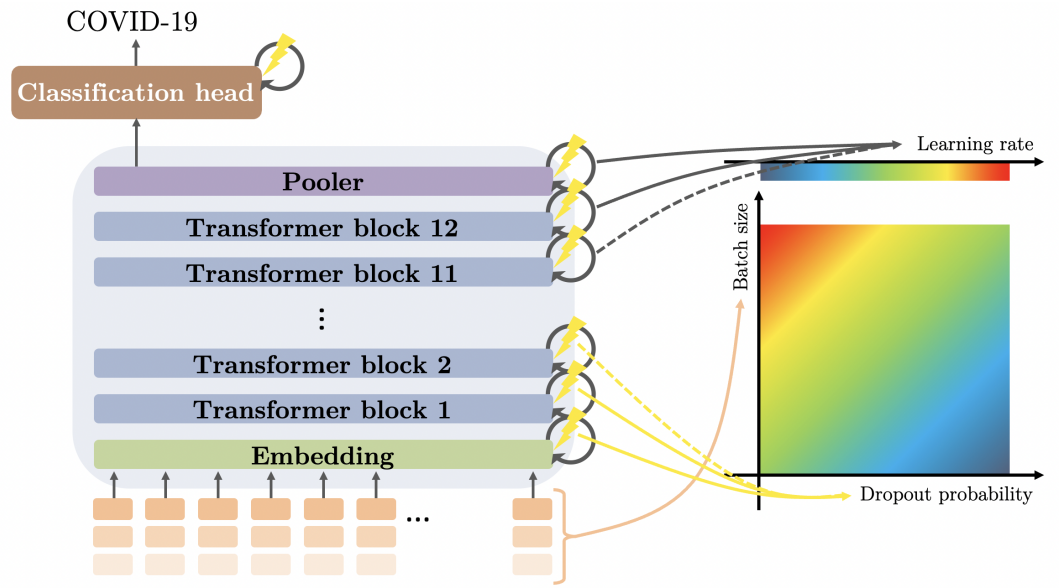
Supplementary Figure 1: All the sections that we find in the radiology reports of our fine-tuning dataset are listed on the y-axis along box plots of their number of tokens on the x-axis. The proportion of reports that include each section is indicated on the left of the section names. Findings represent most of the content, especially in CTs. The rule-based model can drop whenever required the black sections, the yellow Findings section for rare edge-cases and never the red sections which concentrate the most important conclusions of the studies.



Supplementary Figure 2: Four different but compatible strategies to vary the learning rate and/or the momentum across training time and layers.



Supplementary Figure 3: Four distinct unfreezing scenarios regulating the amount of fine-tuning time per layer.



Supplementary Figure 4: We expect that larger batch sizes and higher learning rates allow to prevent overfitting without using as much dropout, while speeding up the convergence of the training.

Implementation

To run our experiments, we implement the models, fine-tuning methods, hyperparameter optimization strategies and XAI visualization tools using primarily PyTorch. The weights of all pre-trained models are downloadable on HuggingFace: we upload a version of the weights of our own pre-training, named **BERT**, on the platform too. Concerning the fine-tuning strategies, in particular ULMFit, we use the fastai library, that provides a *Learner* object that can easily be trained with various unfreezing scenarios or learning rate schedulings, among else. Some object wrappers are needed for fastai to correctly interact with HuggingFace transformers library.

Concerning the hyperparameters optimization algorithms, we implement them using both ray and tune librairies. Ray enables to easily build distributed applications, useful in our case to parallelize the training of models with different hyperparameter settings. Tune gives access to many search algorithms and trial schedulers, such as ASHA, Population-based Training or other state-of-the-art approaches. We also mention the Bayesian Optimization library that we used as a first attempt, but it lacks the diversity and ease of parallelization of Tune.

Once the distributed trainings are launched, we rely on Weights & Biases platform to monitor and visualize the different trials. Finally, Captum helps us visualize the Integrated Gradients outputs: we draw saliency maps that describe which tokens trigger the output of the model on each report.

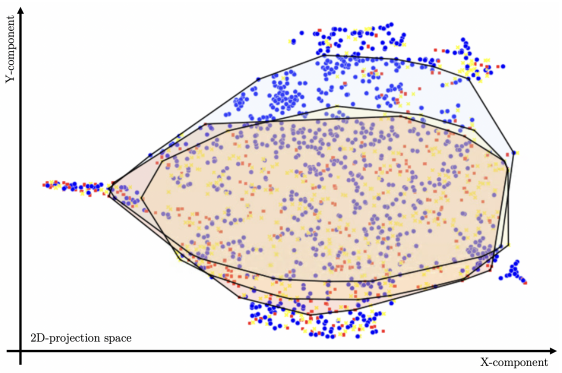
Beyond the release of **BERT** on HuggingFace, the architectures and the code to replicate this work will be made available upon publication, on GitHub.

Supplementary Table 1: Hyperparameter values

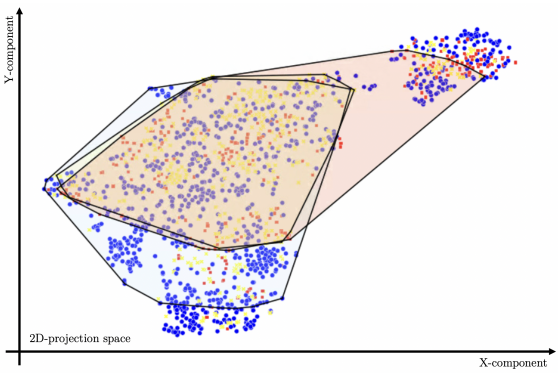
Hyperparameter Name	Exploration Range	Best Model Value	CheXbert Value
Learning Rate	1e-05 to 5e-04	6.29e-05 and 2.06e-05 when training the head only	2e-05
Learning Rate Scheduling	Constant; Triangular; w/ or w/o Final decay	Triangular with final decay	Constant
Momentum	0.8 to 0.95 by 0.05 increments	0.8	Not specified
Momentum Scheduling	Constant; Triangular	Triangular	Constant
Batch Size	8	8	18
Dropout	0 to 0.5 by 0.05/0.1 increments	0.4	0.1
Weight Decay	1e-06 to 1e-04	8.1e-06	Not specified
Number Epochs	1 to 5 by 1 increments	1+3	8
Loss Weights	0 to 0.5 each, summing to 1	∼0.4 except for "No COVID-19"	Not weighted
Head Architecture	1 or 2 linear layers	1 linear layer	14 linear heads
Unfreezing Scenario	4 scenarios	1 epoch head only then full model	Always full model
β, γ's, δ	Various values	Various values	Not used
Optimizer	Adam or AdamW	AdamW	Adam
Hyperparameter Opti.	Tree of P. Esti. or grid	Tree of P. Estimator	Small grid
Hyperparameter Space	Continuous or Discrete	Continuous	Discrete

Transformer hidden-states visualization

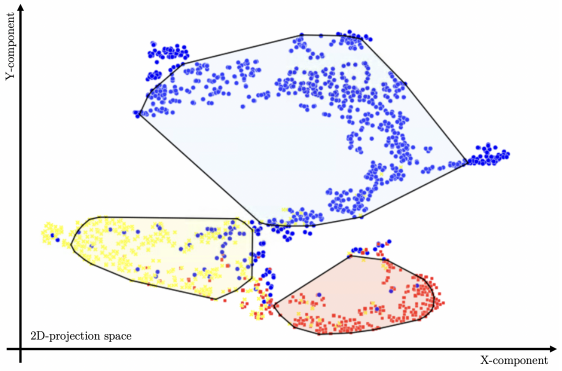
We can assess the performance of the model visually and compare different trained instances. Using a t-SNE projection of the hidden states ([CLS] token only) followed by a k-means clustering, we obtain the visualizations on Figure 5: BERT-base pre-trained only provides very little understanding of the reports to the classification head; **BERT** pre-trained only understands some of the content but do not cluster according to COVID-19 presence, instead it uses other criteria learned during the pre-training; **BERT** fine-tuned clusters almost perfectly per COVID-19 presence and feeds the classification head with well-curated vectors. Besides providing a visual assessment of the performance of different models, this proves how important it is to fine-tune both the classification head and the rest of the model during the fine-tuning phase.



(a) Pre-trained BERT-base



(b) Pre-trained **BERT**



(c) Pre-trained and fine-tuned **BERT**

(d) Network 4

Supplementary Figure 5: t-SNE projection of the CLS hidden tokens of different BERT models, when feeding the COVID-19 reports of the fine-tuning dataset. Each report is represented by a point in the 2-D projection space, and each cluster is represented by a surface encapsulating 75% of the points (the ones that cluster best).

Error Analysis

The COVID-19 test set can be used to understand more the types of errors our model is subject to. The number of false negatives is very limited: only 1.5% of the reports with true *COVID-19* are classified as *no COVID-19*. In addition, after review, we disagree with certain labels assigned to these reports and agree with the classification of the model as *no COVID-19*. To sum up, if we remove the labeling errors, the false negatives of this model are extremely rare.

The uncertain false negatives, ie the misclassifications of true *uncertain COVID-19* as *no COVID-19*, represent 7% of the reports labeled as *uncertain COVID-19*: due to the uncertainty present in these reports, it can be difficult to conclude whether the model is right or wrong, and we consider that these types of errors, which are rare, do not hurt the performance of the model in practice.

A more substantial amount of errors are due to false positives: 1/3 of the total amount of reports are due to true *no COVID-19* reports being classified as *uncertain COVID-19*. The model prefers to have a higher recall and avoid as much as possible false negatives, hence some precautions taken of the reports where it is harder to take a decision.

The rest of the errors are due to misclassifications between *COVID-19* and *uncertain COVID-19* reports. Most of these errors can be detected using the confidence levels that the model provide, along its output: when the confidence levels are lower than usual, we can use a human review to take a decision between *COVID-19* and *uncertain COVID-19*.

In general, we can use confidence levels to filter reports where the model is uncertain, and use human review on them: with this approach, the model can drop the 10% of the reports with lowest confidence and achieve 93.0 macro-averaged F1-score. This process almost eradicates the rare false negatives, with the remaining ones being highly likely mislabeling errors. Using these confidence levels and the XAI plots, we consider that the model is reliable for a clinical use.