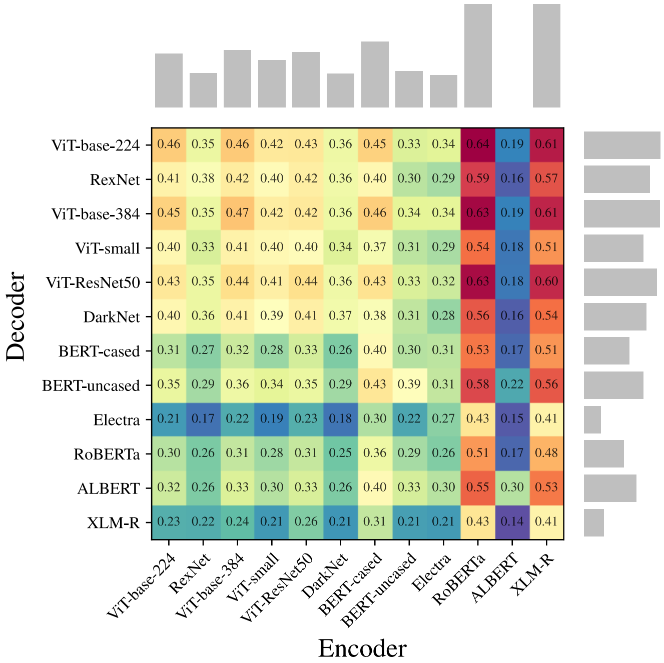
Supplementary Material
Latent Space Translation via Semantic Alignment

Anonymous Author(s)
Affiliation
Address
`email`

1 Additional results

In Figure 1, we present the outcomes of the multimodal experiment with an MLP employed as the classification head. The findings highlight the MLP's capability to leverage cross-modal information, leading to improved performance. However, the underlying mechanisms responsible for this enhancement remain unclear and warrant further investigation.



	ViT-base-224	RexNet	ViT-base-384	ViT-small	ViT-ResNet50	DarkNet	BERT-cased	BERT-uncased	Electra	RoBERTa	ALBERT	XLM-R
ViT-base-224	0.46	0.35	0.46	0.42	0.43	0.36	0.45	0.33	0.34	0.64	0.19	0.61
RexNet	0.41	0.38	0.42	0.40	0.42	0.36	0.40	0.30	0.29	0.59	0.16	0.57
ViT-base-384	0.45	0.35	0.47	0.42	0.42	0.36	0.46	0.34	0.34	0.63	0.19	0.61
ViT-small	0.40	0.33	0.41	0.40	0.40	0.34	0.37	0.31	0.29	0.54	0.18	0.51
ViT-ResNet50	0.43	0.35	0.44	0.41	0.44	0.36	0.43	0.33	0.32	0.63	0.18	0.60
DarkNet	0.40	0.36	0.41	0.39	0.41	0.37	0.38	0.31	0.28	0.56	0.16	0.54
BERT-cased	0.31	0.27	0.32	0.28	0.33	0.26	0.40	0.30	0.31	0.53	0.17	0.51
BERT-uncased	0.35	0.29	0.36	0.34	0.35	0.29	0.43	0.39	0.31	0.58	0.22	0.56
Electra	0.21	0.17	0.22	0.19	0.23	0.18	0.30	0.22	0.27	0.43	0.15	0.41
RoBERTa	0.30	0.26	0.31	0.28	0.31	0.25	0.36	0.29	0.26	0.51	0.17	0.48
ALBERT	0.32	0.26	0.33	0.30	0.33	0.26	0.40	0.33	0.30	0.55	0.30	0.53
XLM-R	0.23	0.22	0.24	0.21	0.26	0.21	0.31	0.21	0.21	0.43	0.14	0.41

	Encoder	Score	Scale
Vision	ViT-base-224	0.46	90.45
	RexNet	0.38	13.46
	ViT-base-384	0.47	89.66
	ViT-small	0.40	50.17
	ViT-ResNet50	0.44	32.10
	DarkNet	0.37	11.62
Text	BERT-cased	0.40	15.43
	BERT-uncased	0.39	14.54
	Electra	0.27	11.94
	RoBERTa	0.51	11.06
	ALBERT	0.30	32.27
	XLM-R	0.41	18.75

Figure 1: Performance comparison between different encoders and data modalities on the `N24News` multimodal dataset. On the right, the accuracy of models trained end-to-end on a single data modality (Score) and their average norm (Scale). On the left the stitching performance between pairs of encoders and decoder. This shows the importance of translating from good encoders, that can even improve unimodal decoder performances. Results obtained with 2000 anchors and `SVD`, with a MLP as classification head.

In Tables 2 and 3 quantitative results for stitching of MLP classifiers (differently from the main manuscript where SVMs are used) trained on top of pre-trained feature extractors, with and without additional L2 normalization, respectively.

In Figures 2 and 3, there are additional reconstruction examples with the same autoencoding setting as in the main manuscript, and with additional L2 normalization, respectively.

In Table 1 more quantitative results for stitching of autoencoders, with added L2 normalization (at training time) to the decoders of the reconstruction setting of the main manuscript.

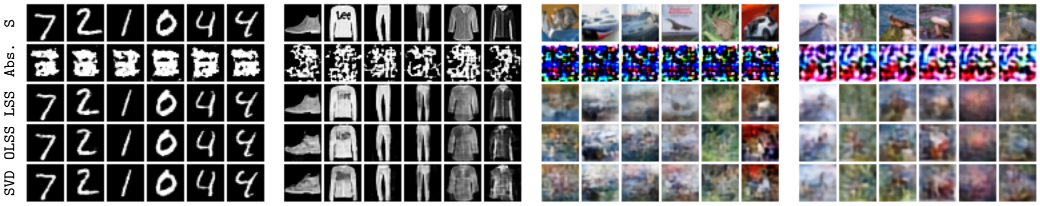


Figure 2: Reconstruction examples grouped by dataset. Each column is a different image, from top to bottom: original image, absolute stitching, `LSS` stitching, `OLSS` stitching, and `SVD` stitching. An L2 normalization is applied to the decoder input.

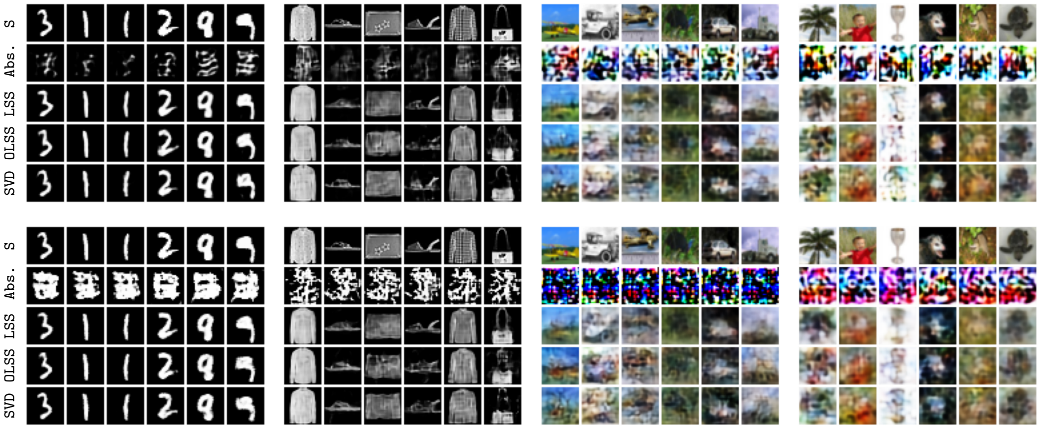


Figure 3: Additional reconstruction examples grouped by dataset. Each column is a different image, from top to bottom: original image, absolute stitching, `LSS` stitching, `OLSS` stitching, and `SVD` stitching. In the first row, no additional normalization is applied on the decoder input; in the second row there is an L2 normalization instead.

Table 1: Zero-shot stitching for generation. With `SVD` for estimating $\mathbf{R}$ and standard scaling as σ. An L2 normalization is applied to the decoder input. We report the latent cosine similarity (*lcos*) and MSE (*lmse*) between the target encoding and the translated one, but also the reconstruction MSE (*rmse*) between the input and the output.

	`MNIST`			`Fashion MNIST`			`CIFAR-10`			`CIFAR-100`		
	lcos	*lmse*	*rmse*	*lcos*	*lmse*	*rmse*	*lcos*	*lmse*	*rmse*	*lcos*	*lmse*	*rmse*
`abs`	0.39	0.98	0.28	0.53	0.97	0.33	0.62	1.23	0.46	0.59	1.17	0.38
`LLS`	0.98	0.17	0.01	0.98	0.18	0.03	0.99	0.16	0.04	0.99	0.13	0.05
`OLLS`	0.89	0.41	0.02	0.91	0.41	0.04	0.96	0.39	0.05	0.93	0.30	0.08
`SVD`	0.97	0.21	0.02	0.97	0.23	0.03	0.99	0.21	0.05	0.96	0.22	0.07

Table 2: Cross-architecture stitching with various methods for estimating $\mathbf{R}$ and employing standard scaling as σ. The stitched decoders are simple MLPs. 5 runs for each encoder-decoder pair. (C) and (F) next to `CIFAR-100` indicate, respectively, coarse-grained and fine-grained.

`dataset`	`original`	`absolute`	`LSS`	`OLSS`	`SVD`
`CIFAR-10`	0.95 ± 0.03	0.16 ± 0.22	0.89 ± 0.11	0.90 ± 0.09	0.93 ± 0.04
`CIFAR100 (C)`	0.82 ± 0.07	0.11 ± 0.21	0.71 ± 0.15	0.74 ± 0.11	0.78 ± 0.07
`CIFAR100 (F)`	0.68 ± 0.13	0.06 ± 0.20	0.55 ± 0.18	0.56 ± 0.17	0.62 ± 0.12
`DBpedia`	0.64 ± 0.18	0.07 ± 0.02	0.53 ± 0.19	0.44 ± 0.21	0.56 ± 0.17
`Fashion MNIST`	0.87 ± 0.02	0.14 ± 0.20	0.83 ± 0.05	0.80 ± 0.06	0.84 ± 0.02
`MNIST`	0.92 ± 0.03	0.15 ± 0.20	0.87 ± 0.08	0.74 ± 0.12	0.88 ± 0.03
`N24News Image`	0.42 ± 0.04	0.08 ± 0.10	0.35 ± 0.07	0.36 ± 0.06	0.39 ± 0.03
`N24News Text`	0.33 ± 0.14	0.05 ± 0.01	0.21 ± 0.13	0.19 ± 0.10	0.34 ± 0.13
`TREC`	0.41 ± 0.07	0.15 ± 0.04	0.37 ± 0.11	0.23 ± 0.08	0.41 ± 0.09

Table 3: Cross-architecture stitching with various methods for estimating $\mathbf{R}$ and employing L2 as normalization. The stitched decoders are simple MLPs. 5 runs for each encoder-decoder pair. (C) and (F) next to `CIFAR-100` indicate, respectively, coarse-grained and fine-grained.

`dataset`	`original`	`absolute`	`LSS`	`OLSS`	`SVD`
`CIFAR-10`	0.95 ± 0.03	0.16 ± 0.22	0.89 ± 0.11	0.89 ± 0.11	0.93 ± 0.04
`CIFAR100 (C)`	0.82 ± 0.07	0.11 ± 0.21	0.71 ± 0.15	0.75 ± 0.13	0.78 ± 0.06
`CIFAR100 (F)`	0.68 ± 0.13	0.06 ± 0.20	0.54 ± 0.18	0.57 ± 0.18	0.61 ± 0.12
`DBpedia`	0.64 ± 0.18	0.07 ± 0.02	0.51 ± 0.18	0.53 ± 0.21	0.49 ± 0.15
`Fashion MNIST`	0.87 ± 0.02	0.14 ± 0.20	0.83 ± 0.05	0.79 ± 0.09	0.84 ± 0.02
`MNIST`	0.92 ± 0.03	0.15 ± 0.20	0.86 ± 0.08	0.80 ± 0.17	0.86 ± 0.04
`N24News Image`	0.42 ± 0.04	0.08 ± 0.10	0.35 ± 0.07	0.36 ± 0.08	0.40 ± 0.04
`N24News Text`	0.33 ± 0.14	0.05 ± 0.01	0.22 ± 0.13	0.26 ± 0.14	0.25 ± 0.16
`TREC`	0.41 ± 0.07	0.15 ± 0.04	0.47 ± 0.13	0.27 ± 0.10	0.49 ± 0.06

1.1 Scale invariance

In this section, we delve into the concept of scale invariance in neural networks and its implications for model compositionality. We start by focusing on the effect of rescaling operations on the latent input encodings and demonstrate that, by construction, certain classifiers exhibit scale-invariance properties without the need for additional priors. Then, by examining the behavior of networks when subjected to a specific type of input manipulation, *rescaling injection*, we aim to demonstrate the robustness and versatility of neural networks in handling different scales of input data. As illustrated in the main manuscript, this is a key advantage in improving the adaptability of our method.

The softmax function, commonly used in neural classifiers, is known to be a temperature-controlled variant of the maximum function:

$$\mathrm{softmax}(x)_i = \frac{e^{\frac{y_i}{T}}}{\sum_j^N e^{\frac{y_j}{T}}} \,. \tag{1}$$

This means that the softmax temperature can be used to control the level of confidence of the classifier's predictions. In this study, we show that a similar effect can also be achieved by rescaling the latent encodings given as input to a trained (and frozen) classifier.

In order to demonstrate this, we first note that the rescaling factor, α, can be factored out of the matrix multiplication in the Linear layers of the classifier. This can be represented mathematically as: $\mathbf{y} = \alpha \mathbf{W}\mathbf{x} + b$, where $\mathbf{x}$ is the input latent encoding, $\mathbf{W}$ is the weight matrix, b is the bias vector, α is the rescaling factor, and $\mathbf{y}$ is the output of the linear layer. This implies that the rescaling operation can be "pushed through" the classifier without affecting its final prediction as it becomes equivalent to some temperature value applied at the softmax level.

Furthermore, we investigate the effect of rescaling when non-linear activation functions are involved and posit that as long as the function has a monotonic interval, if we rescale all the dimensions by an amount similar to the mean scale of the encodings on which the classifier was trained, we end up in the monotonic interval, without losing the scale-invariance property.

In summary, our study provides empirical evidence that neural classifiers that utilize the softmax activation function can, in practice, maintain their scale-invariance properties when the input latent encodings are rescaled. This property is essential to our method, as it allows us to ignore the exact scale when decoding toward an L2-normalized absolute space.

Pre-trained models and scale-invariance We observed that large pre-trained models, such as transformers and resnets, are robust to internal rescaling of the encodings. Although we do not have a strong theoretical explanation for this phenomenon, we hypothesize that normalization layers and the linear separability of the information encoded in the angles instead of the norms may play a significant role. In Figure 4, we demonstrate the invariance a large transformer exhibits when the rescaling injection is applied at different layers: surprisingly, when the rescaling surpasses a certain threshold, the performance difference becomes negligible. These results further emphasize the robustness of these pre-trained models to the rescaling injection and suggest that the scale of the embedding is not a critical factor in their performance.

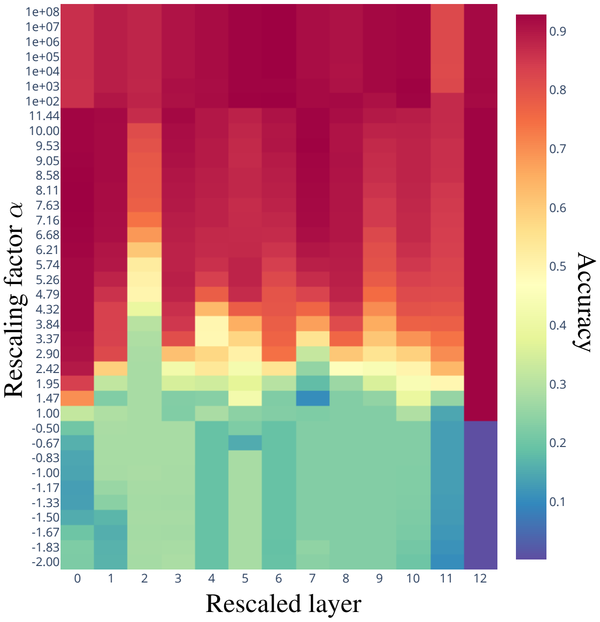


Figure 4: Scale invariance of RoBERTa according to the performance of a downstream classifier trained on the encodings of the last attention layer. At each layer (with 0 being the embedding layer and 12 the output one), one for each run, we rescale the encodings by the specified α and measure its effect on the final accuracy. The performance without any rescaling is 0.92.

Rescale Injection We define the *rescaling injection* as the operation of artificially altering the scale of the features produced at a specific layer of the network. This is achieved by normalizing the embeddings to unit norm and then rescaling them by a factor of α. By varying the value of α, we can observe how the network's performance is affected at different scales. Through this empirical analysis, we aim to provide insight into the scale invariance properties of neural networks and their potential for use in model compositionality.

In Figure 5, we present experimental results investigating the scale invariance properties of neural networks. We trained simple multi-layer perceptrons (MLPs) composed of two hidden layers, with no normalization layers, using encodings produced by the Clip Vision transformer (`clip-vit-base-patch32`) on the `CIFAR-100` (fine) dataset. The MLPs were evaluated using different activation functions: cosine (blue), tanh (orange), and ReLU (green). The rescaling injection technique was applied directly to the input embeddings, rescaling them by α.

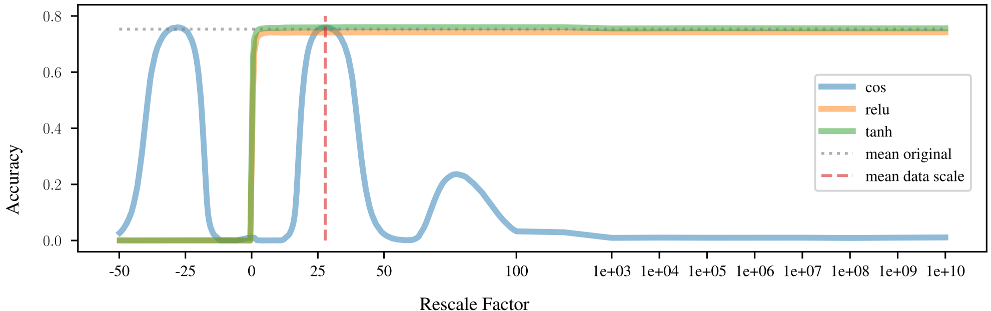


Figure 5: Performance comparison of three Multilayer Perceptrons (MLPs) with different activation functions, namely cosine (blue), ReLU (orange), and tanh (green) at different rescaling factors α. The ReLU and tanh MLPs exhibit scale invariance, while the cosine activation function is only invariant on the mean data scale and its periodic cycles.

We can observe that the scale of the embeddings does not significantly impact the MLPs' performance when using monotone activation functions that do not flip signs. This is a non-trivial result, as the nonlinearity of the activation function, the presence of bias terms b, and the absence of normalization layers make it difficult to predict the effect of an input rescaling on the performance of the network. It is particularly interesting to see that the cosine activation function shows an oscillatory performance, comparable to the original embeddings when rescaled by the mean embeddings scale (vertical red line) or its opposite since it is symmetric.

Our findings indicate that, surprisingly, even the internal layers of large deep learning models exhibit a *positive scale invariance*, as illustrated in Figure 4. The underlying mechanism for this behavior is not straightforward, but we hypothesize that it may result from the interplay between various factors, such as the choice of activation function, the use of normalization layers, the optimization objective and regularization techniques employed during the training phase. Further research is needed to understand and explain this phenomenon fully.

2 Implementation Details

The experiments were conducted using a machine equipped with an Intel Core i7-9700k CPU, 64 GB of RAM, and an NVIDIA 2080TI GPU.

Decoder structure The full implementation details can be found in the attached code. The various experiments can be run by their corresponding notebook, while the source code for the package they are built on can be found under the "src" folder.

- *Autoencoding*. Since the autoencoders were used only on image data, the architecture was a simple sequence of convolutions (in the encoder part) and deconvolutions (in the decoder part). Each interleaved with nonlinear activations.
- *Classification*. The main manuscript refers to "SVM" as the standard SVM implementation in scikit-learn [Pedregosa et al., 2011], with default parameters. The experiments with "MLP" as a classifier refer to a simple stack of 3 linear layers, interleaved by nonlinear activations.

Software and Technologies The research of this study was facilitated by the use of various technologies and tools, which include:

- *NN-Template* [GrokAI, 2021], was used to kick-start the project while also ensuring best practices were adhered to;
- *DVC* [Kuprieiev et al., 2022], was implemented for data versioning;
- *PyTorch Lightning* [Falcon and The PyTorch Lightning team, 2019], contributed to maintaining the integrity of the results and promoting clean, modular code;
- *Weights and Biases* [Biewald, 2020], were employed for logging experiments, running comparisons over extensive sweeps, and sharing models;

- *Transformers by HuggingFace* [Wolf et al., 2020], provided pre-configured transformers for processing both image and text data;
- *Datasets by HuggingFace* [Lhoest et al., 2021], facilitated access to a majority of NLP datasets and ImageNet for computer vision purposes;

Pre-trained encoders All the pre-trained encoders used come from HuggingFace and are listed in Table 4. They are various both in terms of architecture and encoding size.

Table 4: HuggingFace models used as encoders (feature extractors) in the various experiments, with their encoding dimensionality.

Modality	HuggingFace model name	Encoding Dim
Language	bert-base-cased	768
	bert-base-uncased	768
	google/electra-base-discriminator	768
	roberta-base	768
	albert-base-v2	768
	xlm-roberta-base	768
	openai/clip-vit-base-patch32	768
Vision	rexnet_100	1280
	cspdarknet53	768
	vit_small_patch16_224	384
	vit_base_patch16_224	768
	vit_base_patch16_384	768
	vit_base_resnet50_384	768
	openai/clip-vit-base-patch32	768

References

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

GrokAI. nn-template bootstraps pytorch projects by advocating reproducibility & best practices in deep learning, 2021. URL `https://github.com/grok-ai/nn-template`. Software available from https://github.com/grok-ai/.

Ruslan Kuprieiev, skshetry, Dmitry Petrov, Paweł Redzyński, Peter Rowlands, Casper da Costa-Luis, Alexander Schepanovski, Ivan Shcheklein, Batuhan Taskaya, Gao, Jorge Orpinel, David de la Iglesia Castro, Fábio Santos, Aman Sharma, Dave Berenbaum, Zhanibek, Dani Hodovic, daniele, Nikita Kodenko, Andrew Grigorev, Earl, Nabanita Dash, George Vyshnya, Ronan Lamy, maykulkarni, Max Hora, Vera, and Sanidhya Mangal. Dvc: Data version control - git for data & models, 2022. URL `https://doi.org/10.5281/zenodo.7083378`.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. URL `https://github.com/Lightning-AI/lightning`.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://aclanthology.org/2020.emnlp-demos.6`.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. URL `https://aclanthology.org/2021.emnlp-demo.21`.