A. Supplementary Material

In this supplementary material, we provide additional details on downstream tasks, model architecture, and training procedures that were omitted from the main paper due to space constraints.

A.1. Taxonomy of Remote Sensing Foundation Models

The current landscape of RSFMs, as illustrated in Figure 8, is vibrant and rapidly expanding, primarily along two avenues: Visual Foundation Models (VFMs) and Visual Language Models (VLMs). While these models have pushed the boundaries of what is possible in terms of performance, they have largely overlooked the critical dimension of usability, creating a significant bottleneck for the broader scientific community.

VFMs are engineered to interpret a diverse array of visual remote sensing data. Early and influential models like SatMAE [17] and its successors have successfully adapted the masked autoencoder (MAE) [35] paradigm for temporal MSI and RGB imagery, demonstrating strong capabilities in learning spatial patterns. For instance, Scale-MAE [73] introduced a scale-aware pre-training strategy to handle multi-scale geospatial data, while SatMAE++ [67] explored multi-scale reconstructions to better capture scale variability in RS images. To address the need for multi-modal fusion, models like MMEarth [66] and the recently proposed Galileo [92] have emerged, integrating an extensive range of inputs including MSI, SAR, and DEM through sophisticated fusion frameworks. Concurrently, models focused on temporal analysis have also been developed. Presto [91], for instance, is a pixel-level Transformer pre-trained on a global time-series dataset, showing excellent performance in tasks requiring temporal understanding.

However, a unifying weakness across almost all these VFMs is their profound usability challenge. They are released as massive models, placing the onus of data acquisition, complex preprocessing, and computationally intensive fine-tuning squarely on the end-user. This approach requires users to possess significant GPU resources and deep technical expertise, creating a high barrier to entry. While Google's AlphaEarth makes a notable exception by providing pre-generated representation maps, it is not fully open-source.

VLMs for remote sensing aim to bridge the gap between visual data and natural language, partially addressing cross-domain integration. Models like SatCLIP [44], RemoteClip [54], and GeoRSCLIP [123] have adapted contrastive learning (e.g., CLIP [71]) to align satellite images with textual descriptions, enabling powerful zero-shot classification and text-based image retrieval. More advanced models like EarthGPT [121] and SkySenseGPT [57] build upon frameworks like LLaVA [55] to facilitate conversational interaction and multi-sensor (e.g., SAR, infrared) image comprehension. While these models open up exciting new avenues for human-computer interaction with geospatial data, they inherit the same fundamental usability limitations as VFMs. They are typically released as large models requiring expert handling and significant computational power, and none provide large-scale, pre-generated products that would lower the barrier to entry for non-specialist users.

In summary, the field has made tremendous strides in improving the *performance* of RSFMs. Yet, this progress has been largely confined to a model-centric paradigm, inadvertently creating a high barrier to entry that contradicts the goal of democratizing science. The critical gap, therefore, is not a lack of powerful models, but a lack of a user-centric framework that delivers their power in an accessible and ready-to-use format.

A.2. Model Details

Dual-Encoder Architecture Given the distinct nature of Sentinel-1 SAR and Sentinel-2 MSI data, TESSERA employs two separate, parallel transformer-based encoder branches.

- **Sentinel-2 MSI Encoder**: This branch processes a time series of 10 spectral bands (excluding the 60 m bands used for the detection of water vapour and clouds) from Sentinel-2. We used blue (B2), green (B3), red (B4), red edges 1–3 (B5, B6, B7), near-infrared (B8, B8A), and shortwave infrared (B11, B12).
- **Sentinel-1 SAR Encoder**: This branch processes a time series of 2 polarizations from Sentinel-1 (VV and VH).

Each encoder begins by linearly embedding the input features (spectral bands or polarizations) for each time step. To preserve sequence order and incorporate temporal context, learnable positional encodings based on the Day-of-Year (DOY) of each observation are added to these embeddings. The core of each encoder consists of a stack of 4 standard Transformer blocks [93], featuring multi-head self-attention and feed-forward layers to learn temporal patterns within the data streams.

To derive a single vector summarizing the entire time series for each modality, an attention-pooling layer weighs the importance of different time steps before aggregation. The resulting modality-specific embeddings (one from the S1 encoder, one from the S2 encoder) are then fused using a multi-layer perceptron.

Projector Network The fused embedding from the dual-encoder stage is subsequently fed into a large projector network. This projector is a six-layer MLP. Its architecture comprises an input layer mapping the fused embeddings to 16,384 dimensions, followed by four hidden layers of 16,384 dimensions each, and a final linear output layer. Each of the first five layers is a fully-connected linear

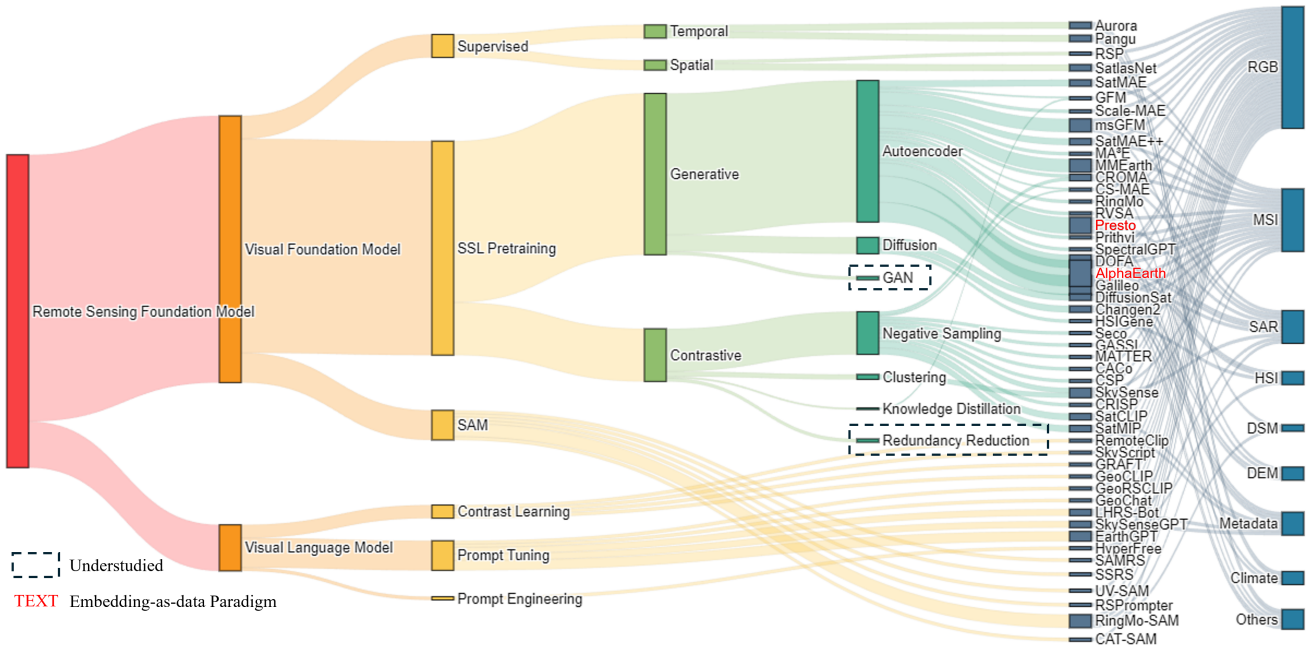

Figure 8. Taxonomy of RSFMs. All RSFMs are categorized into two primary classes: visual foundation models and visual language models, which are further subdivided according to different methodologies. The final column of the Sankey diagram illustrates the data modalities used by each foundation model. Dashed rectangles indicate areas with minimal or virtually no existing research. The checkerboard pattern denotes that, in addition to providing model parameters, these foundation models also offer globally generated, analysis-ready representation products derived from the model itself.

layer followed by Batch Normalization and a ReLU non-linear activation function, making the network deep and highly non-linear. This significant expansion in dimensionality is crucial, as suggested by the original Barlow Twins work [116], to enable effective redundancy reduction during the self-supervised loss computation. The final output of the projector is then used in the loss calculation. For downstream tasks, we used the 128-dimensional output from the fusion MLP (before the projector) as the final pixel embedding. The TESSERA encoder (up to the fusion MLP) has approximately 46 million parameters, while the projector accounts for the majority of the model's $\sim$1.4 billion parameters.

Table 4. **Architectural parameters of the TESSERA v1.0 model.**

Component	Parameters	Percentage of Total
Encoder (Feature Extractor)	**45,697,026**	**3.29%**
Sentinel-1 Encoder	22,846,465	1.64%
Sentinel-2 Encoder	22,850,561	1.64%
Fusion MLP	**131,200**	**<0.01%**
Projector (for Training only)	**1,344,700,421**	**96.71%**
Total Parameters	**1,390,528,647**	**100.00%**

Self-Supervised Training The TESSERA model is trained using a modified Barlow Twins objective function [116]. For this objective, two distorted views, denoted as Y_A and Y_B, are generated for each input d-pixel. In TESSERA, these views are created by independently running the temporal sampling and preprocessing pipeline twice for the Sentinel-1 and Sentinel-2 data associated with a given d-pixel. This process involves:

1. For each view, independent sampling of a fixed number of valid observation dates from the annual Sentinel-2 time series (10 spectral bands).
2. For each view, independent sampling of a fixed number of valid observation dates from the annual Sentinel-1 time series (2 polarizations).

These views represent different, valid, but inherently incomplete glimpses of the pixel's true temporal-spectral evolution, akin to observing the same location through intermittent cloud cover or from different satellite passes at different times. The model learns by reconciling these partial views. The inherent differences between the Sentinel-1 SAR and Sentinel-2 MSI modalities further provide diverse perspectives on the same underlying physical processes. Thus, our augmentations are fundamentally about sampling from the available, inherently incomplete information streams, rather than artificially distorting a complete input.

A.3. Pre-training Details

The TESSERA model was pre-trained using approximately 800 million d-pixels drawn from 3,012 globally distributed Military Grid Reference System (MGRS) tiles chosen from the years 2017-2024. d-pixels were generated by spatially downsampling Sentinel-1 and Sentinel-2 data by a factor of 400. This was achieved by systematically taking every 20th pixel along the horizontal and vertical axes. Each d-pixel contained the annual time series for 10 spectral bands of Sentinel-2 data and the annual time series of VV and VH polarizations for Sentinel-1 data.

For each pixel location, after removing invalid observations (e.g., due to cloud cover for Sentinel-2), we performed sparse temporal sampling. This involves randomly selecting a fixed number of 40 valid observation dates from the year's data. The sequence length for the Transformer encoders was fixed at 40 timesteps for both modalities. If there are fewer than 40 cloud-free dates, replacement sampling is performed. This strategy standardizes the input sequence length and serves as a key data augmentation mechanism, building invariance to data gaps and teaching the model that the underlying signal persists regardless of the specific dates observed. Finally, these values were standardized using global statistics to stabilize training. The temporal context of each sampled observation was encoded by transforming its normalized Day-of-Year (DOY) into sine and cosine features, which were then concatenated with the corresponding spectral or backscatter measurements.

As we were in a compute-limited rather than data-limited pre-training regime, we trained for a single epoch, which corresponded to approximately 6200 GPU hours on 16 AMD MI300X GPUs (192GB memory each). We used PyTorch with Fully Sharded Data Parallel (FSDP) and Automatic Mixed Precision (AMP) enabled. AdamW optimizer was used with a base learning rate of 0.002 and weight decay of 1×10^{-6}. The learning rate schedule consisted of a linear warmup during the initial 10% of steps, followed immediately by a cosine decay for the remainder of training. The global batch size was 32,768. Key training dynamics, including loss curves, learning rate schedule, and evolution of downstream performance during pre-training, are visualized in Supplementary Figure 10.

A crucial aspect of our training methodology is a data shuffling strategy, essential for learning globally representative features from a representative sample of a vast and geographically diverse dataset. Given that d-pixels within an individual MGRS tile exhibit high spatial autocorrelation, a naive sequential or locally-shuffled data loading process would expose the model to strong geographic biases in each batch. To overcome this, we developed a custom data processing pipeline to implement a truly global shuffle across all ≈800 million training samples, which constituted more than 2 TB of initial d-pixel data. This pipeline is conceptually illustrated in Figure 11 a.

The impact of this approach on training stability is empirically demonstrated in Figure 11 b. Compared to a conventional localized shuffling strategy, which results in a highly volatile loss curve (top plot), our global shuffling strategy yields a markedly smoother and more stable convergence (bottom plot). This enhanced stability is fundamental for robust convergence and for preventing the model from overfitting to regional characteristics.

Operationally, the process begins with the aggregation of d-pixels from all MGRS tiles into a single, comprehensive pool. A global shuffling operation is performed on this pool, a critical step to break the spatial contiguity of data from individual tiles and ensure each training batch contains a diverse mix of geographical and environmental contexts. Following this global shuffle, the data augmentation required by the Barlow Twins framework is applied.

To manage the significant I/O demands of shuffling and augmenting such a large volume of data, we developed a custom high-performance pipeline that handles the reading of raw d-pixel data, executes the global shuffle, and prepares the data for augmentation. The resulting pairs of augmented d-pixels are then serialized into a compact, pickle-like file format. These files are organized into manageable chunks and loaded by PyTorch *DataLoader* workers, which stream the data and assemble the final training batches. This end-to-end pipeline ensures that each batch presented to the model is a well-shuffled, globally diverse representation of Earth's surface characteristics, which is fundamental for training a robust pixel-wise foundation model like TESSERA.

A.4. Global Embedding Map Generation

A primary output of the TESSERA project is the generation of annual global embedding maps with 10 meter resolution for the years 2017-2024. To generate these maps, the pre-trained and frozen TESSERA dual encoder (excluding the projector) is used. For each 10 meter pixel on the globe and for each year:

1. The full Sentinel-1 and Sentinel-2 time series data at 10 meter resolution are acquired and pre-processed to form d-pixels. Unlike pre-training, no spatial downsampling is performed at this stage.
2. A fixed number of 40 timesteps is sampled from the valid observations within the year for both Sentinel-1 and Sentinel-2 data, along with their DOY positional encodings.
3. These sampled time series are fed into their respective frozen TESSERA encoders.
4. The outputs from the S1 and S2 encoders are fused by the MLP, producing a 128-dimensional embedding vector for that pixel for that year.

This process is repeated for all land pixels globally to create

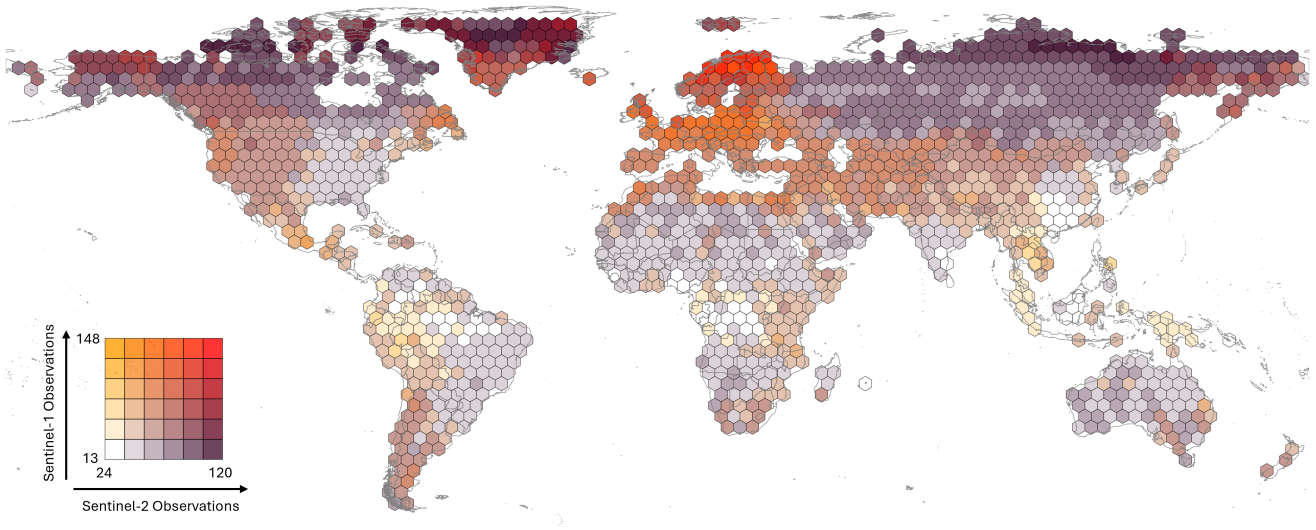

Figure 9. **TESSERA is trained on globally-distributed training data.** TESSERA was trained on over 3000 MGRS tiles distributed globally from 2017 to 2024. The colour of each hexagon in the map corresponds to the average number of valid observation days for Sentinel-1 (S1) and Sentinel-2 (S2), as defined by the bivariate colour legend. This visualization highlights the density of combined S1 and S2 observations available for training across different regions, ensuring the model learns from a diverse range of geographical and environmental conditions.

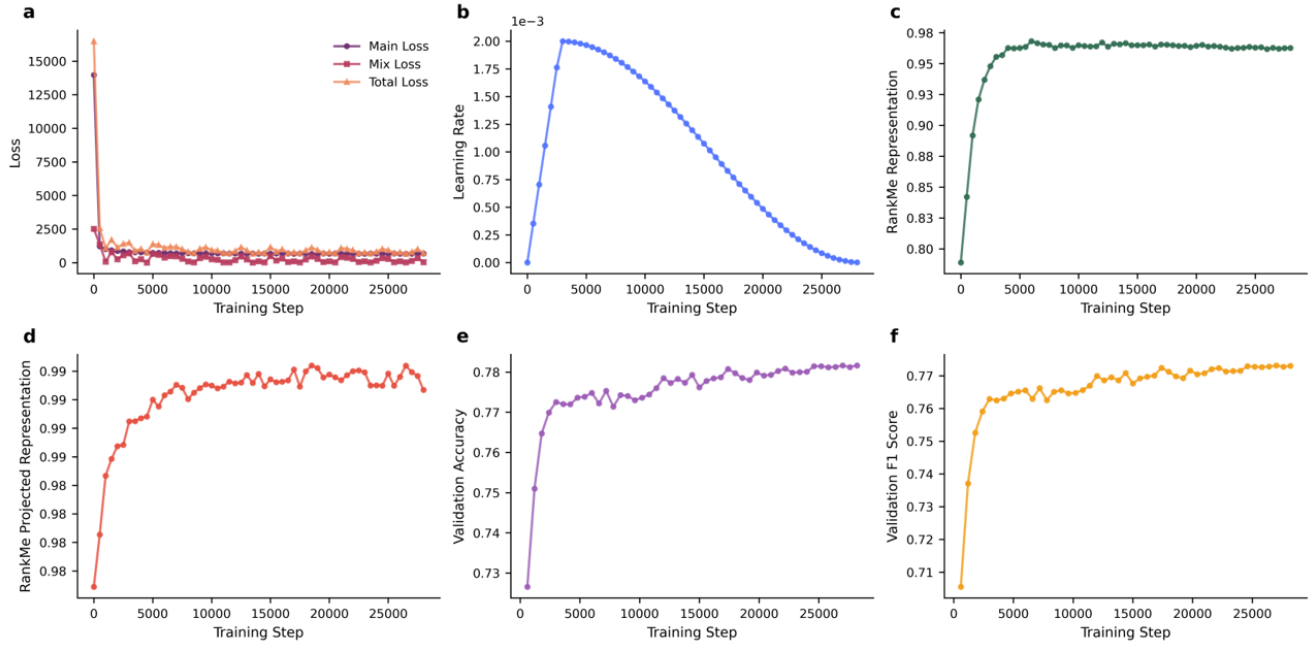

Figure 10. **Training dynamics curves show that TESSERA rapidly reaches its asymptotes.** **a**, Training loss curves, showing the main Barlow Twins loss, the mixup regularization loss, and the total loss. **b**, The learning rate schedule, consisting of a linear warmup phase followed by a cosine decay. **c**, RankMe score calculated on the high-dimensional (16,384-D) output of the projector, which quickly saturates, indicating the model is effectively utilizing the large feature space for redundancy reduction. **d**, RankMe score of the final 128-D fused representation, showing that the embeddings used for downstream tasks become progressively richer and less redundant. **e**, **f**, Downstream performance on the Austrian crop classification validation set, tracked by accuracy and F1 score respectively during pre-training, showing a steady improvement that correlates with the training progress.

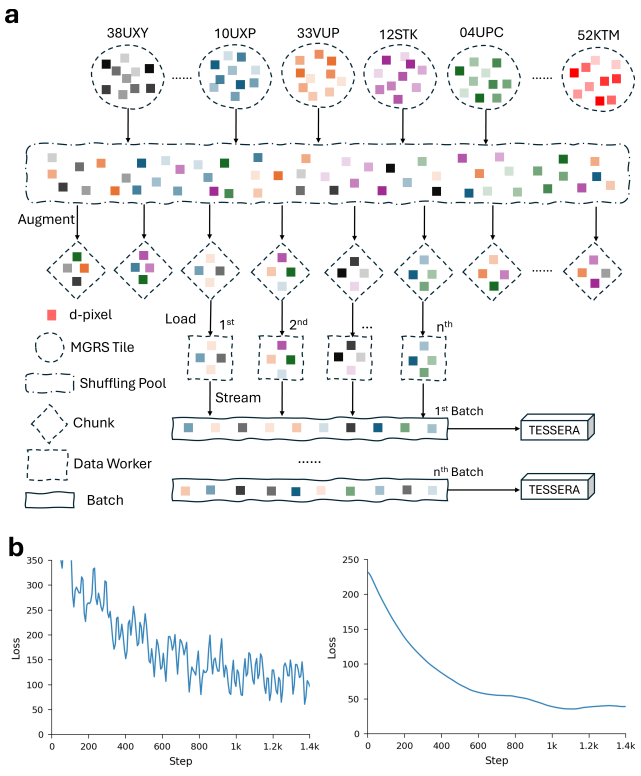

Figure 11. **Data shuffling greatly improves training stability.** **a**, Schematic of the data shuffling and loading process. D-pixels (coloured squares) from thousands of MGRS tiles are first aggregated into a global pool. A custom Rust binary performs a global shuffle on this multi-terabyte dataset before applying augmentations. The processed data is then organized into chunks and streamed by data workers to form well-shuffled, globally diverse training batches. **b**, Comparison of training loss curves. The top plot shows the volatile loss progression typical of a localized shuffling strategy, which is susceptible to geographic bias. The bottom plot shows the significantly smoother and more stable loss curve achieved with our global shuffling pipeline, demonstrating more effective and robust model convergence.

an annual embeddings map of shape (H, W, 128), where H and W are the dimensions of the global 10 meter grid.

A.5. Spatial Context: Pixel-wise vs. Patch-based Approaches

In this section, we elaborate on the rationale behind TESSERA's pixel-wise design compared to traditional patch-based architectures. By maintaining pixel-wise independence, TESSERA avoids embedding fixed spatial priors, which preserves the integrity of raw spectral signatures and ensures downstream performance is driven by fundamental signal characteristics rather than pre-determined structural assumptions.

Introducing spatial context via fixed patches (Spatial variant) introduces significant vulnerabilities, particularly in regions with high cloud frequency such as Borneo. As the quantitative comparisons demonstrate (Figure 12a), introducing patch-based spatial priors can negatively impact the robustness of the representations. Furthermore, filtering for strictly clear spatial patches results in an exponential decrease in data availability, leading to severe embedding artifacts. This vulnerability is clearly visible in the embedding visualizations of the Borneo region (Figure 12b), where imposing a spatial patch requirement drastically reduces valid data and creates massive spatial voids. By extracting features at the pixel level, TESSERA allows researchers to incorporate domain-specific spatial constraints *post hoc* without compromising the foundational embeddings.

a) Budget = 100 PFLOPS Repeat Times: A 10 P 5 B 20

TESSERA	Input Size	A F1↑	P mIoU↑	B RMSE↓
Current	1×1	**71.72**±0.35	36.92±0.29	**15.45**±0.73
Spatial	3×3	71.09±0.40	36.86±0.16	16.02±1.18
Spatial	15×15	71.34±0.24	**37.33**±0.10	18.25±1.91

A Austrian Crop (30%) P PASTIS-R (All) B Borneo CHM (All)

b) Year: 2017 Region: Borneo

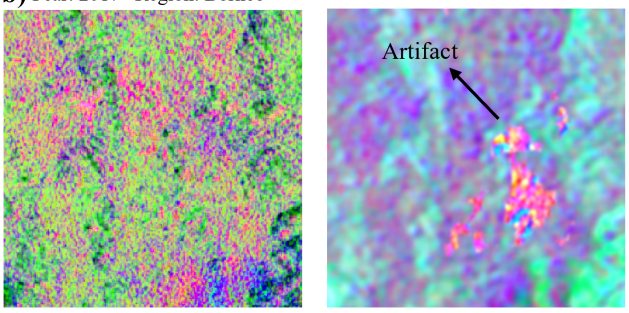

Figure 12. **Spatial Context Ablation.** (a) Quantitative comparison detailing performance differences when introducing patch-based spatial priors. (b) Visualization of the generated embeddings in the Borneo region.

A.6. Geographic Generalization

To demonstrate TESSERA's ability to generalize to diverse phenological regions and smallholder agricultural settings, we evaluate our model on the CropHarvest Togo dataset [90] and the Ethiopia Crop classification dataset [10]. Despite using only a continuous DoY sinusoidal encoding as a weak temporal prior, TESSERA correctly handles local seasonality driven primarily by spectral trends.

Table 5 compares TESSERA against other baseline foundation models. TESSERA significantly outperforms CROMA and AlphaEarth on both benchmarks, demonstrating superior generalization in challenging zero-shot and low-data transfer scenarios characteristic of Sub-Saharan Africa.

Table 5. **Downstream Performance on Sub-Saharan African Benchmarks.** Evaluation using F1 scores (%) for the Togo (CropHarvest) and Accuary (%) for the Ethiopia Crop dataset. TESSERA demonstrates superior adaptability to local seasonalities compared to existing models.

Model	Togo (F1 %)	Ethiopia (Acc %)
CROMA	79.15	/
AlphaEarth	/	∼41
TESSERA (Ours)	**83.01**	**44.92**

A.7. Scaling with data and network size

To identify an optimal and efficient architecture for TESSERA, and to understand the scaling properties of our model, we conducted a series of experiments using the Austrian crop classification task as the downstream task. Our investigation focused on two main areas: the impact of the projector network's dimensions and the overall model scaling laws concerning performance versus computational cost (Figure 7).

Our experiments revealed that the performance of the final embeddings on the chosen downstream task of crop classification in Austria was relatively insensitive to the size of the Transformer encoders. We found that using 4 transformer layers, each with 4 attention heads, provided a strong baseline without incurring excessive computational overhead during inference. This makes the feature-extracting part of the model computationally efficient.

In contrast, the architecture of the projector network proved to be critical for effective self-supervised learning. As detailed in Section A.2, the full projector is a six-layer MLP, consisting of an input and an output layer for accommodating dimensional changes, and four core hidden layers. Our parameter sweep focused on this computationally intensive hidden block. We varied the number of these hidden layers (referred to as 'depth' in Figure 7 a) and their width (number of neurons per layer), with results shown in Figure 7 a. As discussed by Zbontar et al. [116], deeper and wider projectors consistently improved performance, as measured by the validation F1 score in the downstream task. The highest validation score was achieved with the largest configuration tested (8 hidden layers, 32,768 neurons wide). However, this configuration led to an out-of-memory error with our standard global batch size; therefore, we halved the batch size to 16,384 to enable stable training for this specific setup, indicated by the dagger symbol (†) in the figure. For the final TESSERA v1.0 model, we selected the configuration with four hidden layers and a width of 16,384, marked with a star (*), as it provided a strong balance between high performance and computational efficiency.

Furthermore, we analysed the relationship between model performance, the number of trainable parameters, and the total training computation (FLOPs), as shown in Figure 7 b. The results demonstrate a clear scaling trend: model performance improves predictably with increases in model size and computational budget. This confirms that TESSERA's performance can be further enhanced with greater computational resources, following established scaling laws for large neural networks. The final TESSERA v1.0 model is marked with a star (*) on this curve, highlighting its position as a model that balances state-of-the-art performance with cost-effective training requirements.

A.7.1. Ablation study details

To validate the contribution of each key component within the TESSERA framework, we performed an ablation study. We systematically removed or disabled individual components from our full model (the baseline) and evaluated the impact on the 100x downsampled Austrian crop classification dataset.

Performance was evaluated using two metrics: the standard validation overall F1 score, and a measure of the effective rank of the learned embeddings, which we refer to as the RankMe score [32]. The RankMe score quantifies the richness of the embeddings by measuring how uniformly the information is distributed across the feature dimensions. Given a batch of embeddings represented by the matrix $Z \in \mathbb{R}^{N \times D}$, where N is the batch size and D is the embedding dimension, we first compute its singular values $s_1, s_2, \ldots, s_D$. These are normalized to form a probability distribution $p_i = s_i / \sum_j s_j$. The RankMe score is the Shannon entropy of this distribution, normalized by the maximum possible entropy:

$$\mathcal{R}_{\text{RankMe}} = \frac{-\sum_{i=1}^{D} p_i \log(p_i)}{\log(D)} \tag{1}$$

A score closer to 1 indicates that the embeddings are of higher effective rank, utilizing the full dimensionality of the feature space more effectively to create richer, less redundant representations.

Our analysis reveals several key insights into the model's architecture and training strategy. The most critical component turns out to be the global data shuffling pipeline, which is designed to break spatial autocorrelation and present the model with geographically diverse mini-batches. Removing mixup regularization also severely degrades results. When both are removed, the model's performance collapses, showing that the combination of global shuffling and mixup regularization is needed to prevent overfitting and to learn generalizable robust representations.

With removal of Sentinel-1 data, task performance decreases moderately, demonstrating some gain from the fusion of optical and SAR data yields. However, performance is still relatively good even with only Sentinel-2 data. This highlights both TESSERA's overall robustness and its

Table 6. Parameter sweep of the number of sampled timesteps (L). We evaluate the trade-off between F1-score on the Austrian crop classification dataset and inference speed.

L	Inference Speed (samples/sec)	Encoder Param.	Training Samples	F1 Score (%)
15	1509	∼36.4M	22M	65.04
20	1452			66.58
30	1276			71.43
40	**1068**			**73.34**
50	724			73.69
60	497			73.77

continued efficacy in regions or historical periods where Sentinel-1 data are sparse or unavailable, which is not uncommon in practice.

Finally, the effect of quantization is particularly noteworthy. We employ Quantization-Aware Training (QAT) to convert the embeddings from 32-bit floating-point (float32) to 8-bit integer (int8) format. Removing this quantization step (i.e. using full float32 precision) results in a marginal performance improvement. This demonstrates the success of our QAT strategy: we achieve a nearly four-fold reduction in the storage and bandwidth requirements for our final data products, with a negligible impact on embedding quality. This underlies our "Embeddings-as-Data" approach, making global-scale high-resolution analysis practical: Without this, each year's embeddings would consume 1PB, making the hosting of 8 years of embedding costly and likely unachievable.

A.7.2. Justification for the Number of Sampled Timesteps (L)

As detailed in the main paper, TESSERA is trained using sparse random temporal sampling to enforce invariance to the selection of valid observations. A critical hyperparameter in this process is L, the number of valid timesteps sampled. We finalized our choice at $L = 40$. This decision is supported by two primary justifications:

1. **Performance vs. Efficiency Trade-off:** We conducted a comprehensive parameter sweep on the Austrian crop classification dataset to determine the optimal balance between model performance and computational cost (i.e., inference speed). As demonstrated in Table 6, setting $L = 40$ achieves the best trade-off. While further increasing L (e.g., to 50 or 60) results in only marginal or negligible performance gains, it increases the computational overhead during inference.
2. **Global Data Representativeness:** Our choice is also empirically grounded in global-scale data statistics. We analyzed the global distribution of valid Sentinel-2 observations acquired between 2017 and 2024 and found that the global average number of valid observations per pixel is approximately 37.6. This confirms that $L = 40$ is not only efficient but also a robust and representative value for global-scale modeling, closely aligning with the average valid Sentinel-2 data availability worldwide (see Figure 13).

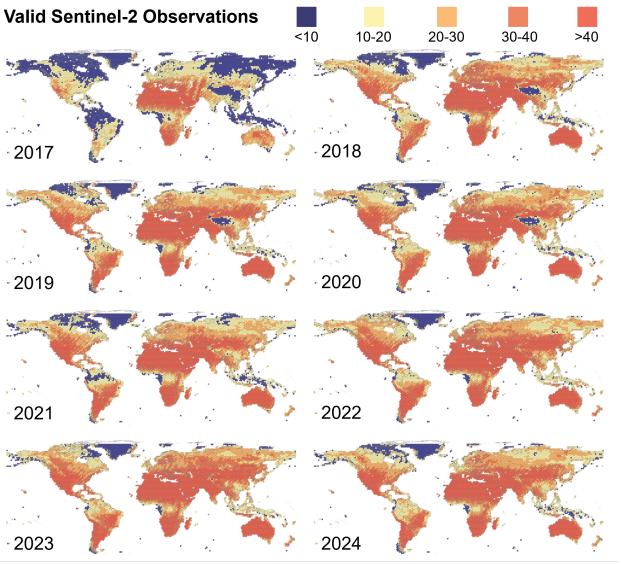

Figure 13. A distribution map of the number of valid Sentinel-2 observations from 2017 to 2024. Valid observations are defined as those that are free of clouds, water bodies, and snow.

Furthermore, we observed that downstream task performance is largely insensitive to the number of Sentinel-1 observations sampled, likely due to the higher temporal density and all-weather capabilities of SAR. Therefore, for the sake of architectural consistency and to simplify the multimodal training pipeline, we also set the number of sampled timesteps for Sentinel-1 to $L = 40$.

A.8. Downstream Task Application Methodology

A.8.1. TESSERA

A core motivation for self-supervised learning with foundation models is the creation of task-agnostic feature embeddings that can be effectively transferred to various downstream tasks, particularly in scenarios with limited labelled data. Having pre-trained TESSERA on large unlabelled datasets, we evaluated the utility of its learned embeddings across different remote sensing applications. The evaluation methodology involves using pre-trained TESSERA encoders as fixed feature extractors.

1. **Download Embeddings for the Region of Interest**: The GeoTessera Python library allows users to download embeddings for a desired region and year in the form of a numpy array.
2. **Prepare labelled Downstream Data**: The labelled dataset for the target task (e.g., pixel-level crop-type labels, canopy height measurements, or land use change polygons) is prepared.
3. **Design Task-Specific Head**: A lightweight, task-specific neural network module (the "head") is designed.

This head takes the extracted TESSERA embeddings as input.
- For pixel-wise classification (e.g., crop classification), the head is typically a shallow MLP (1-3 layers) ending in a softmax output layer.
- For pixel-wise regression (e.g., canopy height regression), the head is usually an MLP ending in a single linear output neuron.
- For tasks requiring spatial context from the embeddings (e.g., canopy height mapping over an area, semantic segmentation), the input to the head can be a patch of TESSERA embeddings (e.g., $64 \times 64 \times 128$). The head might then be a convolutional architecture, such as a UNet, that processes these spatial feature maps to produce dense predictions.

4. **Train Downstream Head**: Only the parameters of this newly defined task head are trained using the extracted TESSERA embeddings as input features and the corresponding labels. Standard supervised learning techniques, optimizers (e.g., Adam), and task-appropriate loss functions (e.g., Cross-Entropy for classification, Mean Squared Error for regression) are used. This training typically requires significantly less labelled data and computational power compared to training a deep model from scratch.
5. **Evaluation**: Once the head is trained, inference is performed on a test set by extracting TESSERA embeddings for the test samples and passing them through the trained head. Performance is evaluated using standard metrics relevant to the task.

This workflow allows the use of TESSERA embeddings in a range of diverse applications, demonstrating its role as a foundational model for geospatial analysis.

TESSERA users who would like to generate their own embeddings can load the frozen weights from the saved pre-training checkpoint into the TESSERA dual encoder architecture. For each input sample (e.g., pixel, object, or patch) in the region of interest, its corresponding Sentinel-1 and Sentinel-2 time series data for the relevant year must be downloaded and processed using the d-pixel creation pipeline. These d-pixels are passed through the frozen encoders and their outputs are fused by the MLP to generate the final 128-dimensional TESSERA embedding for that sample. To enhance stability, multiple embeddings can be created with different random temporal samples and averaged.

A.8.2. Presto & AlphaEarth

For Presto and AlphaEarth, we adopted a methodology conceptually similar to TESSERA's, focusing on evaluating their publicly available embeddings. For Presto, we used the official Google Earth Engine (GEE) scripts provided by the authors to generate embeddings for the regions of interest. The data preprocessing was executed on GEE, and inference was subsequently performed on Google Vertex AI, loading the officially released pre-trained model weights. For AlphaEarth, we directly used the publicly available Google AlphaEarth Satellite Embedding V1 dataset [13]. In both cases, the pre-computed embeddings served as the input features. The visualization of embeddings for some downstream tasks of TESSERA, AlphaEarth, and Presto are shown in Figure 14. We then designed and trained lightweight task-specific heads (e.g., MLPs) for each downstream task, following the same principle as described for TESSERA.

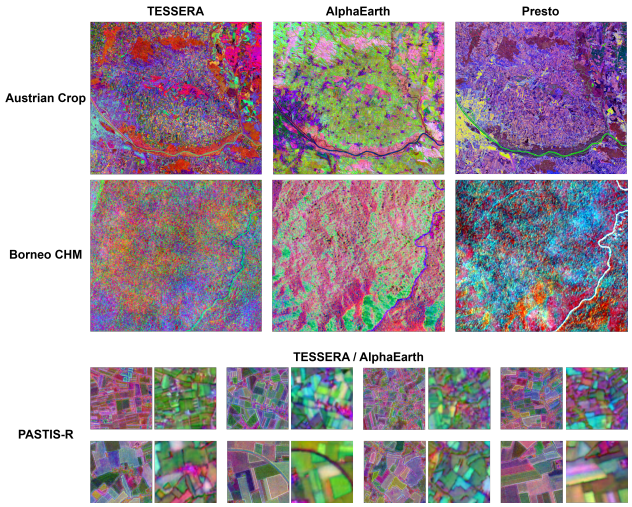

Figure 14. The visualization of embeddings for some downstream tasks of TESSERA, AlphaEarth, and Presto

A.8.3. Other RSFMs

To evaluate the performance of other RSFMs, we used the standardized *Pangaea* evaluation framework. This framework provides a consistent protocol for benchmarking RSFMs across a suite of diverse downstream tasks. We evaluated these models on the Austrian Crop (Segmentation), PASTIS-R, Biomassters, and Borneo CHM datasets. Unlike our embedding-focused approach for TESSERA, the Pangaea protocol typically involves training task-specific decoders on top of the pre-trained encoder features. For classification tasks, a simple `LinearClassifier` head was attached and trained. For the regression and segmentation tasks, heavier decoders such as `UperNet+LTAE` or a baseline `Unet` were employed, as specified in the benchmark's guidelines. The exact downstream task heads used for these models are detailed in Table 7.

A.9. Downstream Task: Austrian Crop Classification

Dataset and Preprocessing We used the publicly available INVEKOS dataset for Austria, focusing on the 2022

Table 7. **Downstream task head specification.** This table lists the decoders (heads) used for different tasks across all models. Note that for each model listed, the Regression and Segmentation tasks use the same head.

Model	Classification Head	Regression & Segmentation Head
CROMA	LinearClassifier	UperNet+LTAE
DOFA	LinearClassifier	UperNet+LTAE
Prithvi	LinearClassifier	UperNet+LTAE
RemoteCLIP	LinearClassifier	UperNet+LTAE
SatlasNet	LinearClassifier	UperNet+LTAE
Scale-MAE	LinearClassifier	UperNet+LTAE
SpectralGPT	LinearClassifier	UperNet+LTAE
Galileo	LinearClassifier	UperNet+LTAE
Skysense	LinearClassifier	UperNet+LTAE
ViT Baseline	LinearClassifier	UperNet+LTAE
UNet Baseline	LinearClassifier	Unet
Presto	LinearClassifier	Unet
AlphaEarth	LinearClassifier	Unet
TESSERA	LinearClassifier	Unet

growing season [1]. The data set originally contained 154 crop types, which we grouped into 17 broader classes (for example, combining sugar beet and feed beet) based on phenological similarity and sample availability to ensure robust training and evaluation.

When constructing the pixel-level classification dataset, we use fields as the unit for data splitting. Each field is assigned a unique identifier along with its corresponding area. We allocate pixels from fields constituting X% of the total area to the training set, while the remaining (1-X)% of the total field area is divided such that 1/7 is used for validation and the rest for testing. For training, validation, and test splits, we ensure that all 17 crop types are represented in each split.

For the patch-based dataset designed for segmentation tasks, we first divide the region of interest vertically into five equal parts. From the topmost section, we extract 1000 patches of 640m × 640m as the test set. An equal number of patches with the same dimensions are cropped from the section immediately below for the validation set. From the remaining three sections, 3000 patches of the same size are sampled to form the training set. All patches maintain a spatial resolution of 10 meters.

Pixel-wise Classification Baselines To provide a comprehensive performance comparison, we implemented three distinct models for pixel-wise classification:

- **TESSERA + MLP**: The primary model, where the frozen 128-dimensional TESSERA embeddings were used as input to a simple MLP. The MLP consisted of two hidden layers with 256 and 128 neurons, respectively, using ReLU activation functions, followed by a softmax output layer for the 17 classes.
- **Presto + MLP**: For a direct and fair comparison with a leading foundation model, we used the official pre-trained Presto model [91] to generate its pixel embeddings. These embeddings were then fed into an MLP head identical to the one used for TESSERA.
- **AlphaEarth + MLP**: To include another key foundation model, we used the publicly available Google AphaEarth Satellite Embedding V1 dataset [13]. These embeddings were similarly fed into an MLP head identical to the one used for TESSERA.
- **Random Forest**: As a traditional baseline, we trained a Random Forest (RF) classifier directly on raw time-series data. For each pixel, we used all available Sentinel-1 (2 polarizations) and Sentinel-2 (10 spectral bands) observations throughout the year. The temporal and spectral/polarization dimensions were flattened and concatenated to form a single 1256-dimensional feature vector. The RF model was configured with 100 trees, while all other hyperparameters retained their default values as implemented in the SCIKIT-LEARN library.

For the experiments shown in the main text's figure, we trained these models on randomly selected subsets of the data (from 1% to 30% for panel a; specified samples-per-class for panel b), using a fixed validation set for hyperparameter tuning and a held-out test set for final evaluation.

Patch-wise Semantic Segmentation To assess spatial-contextual performance, we conducted a semantic segmentation experiment. The approach varied based on the foundation model's architecture:

- For the foundation models capable of generating pixel-wise embeddings, namely **TESSERA**, **Presto** [91], and **AlphaEarth** [13], a distinct workflow was used. We first construct patches from their pixel-level embeddings (e.g., a data cube of size $H \times W \times C$). These patches were then fed into a standard **UNet** architecture [75], which learns to model the spatial relationships between the embeddings to produce a segmentation map. It is noteworthy that while TESSERA and Presto's base embeddings do not explicitly model spatial context, AlphaEarth's do. However, their shared ability to generate pixel-level outputs makes them suitable for this UNet-based downstream approach.
- For other foundation models that are inherently patch-based (e.g., Prithvi [84], Satlas [8]), their encoders already process image patches. Therefore, for these models, we only attach and train a **UPerNet decoder head** to their encoders to generate the final segmentation output. We did not freeze the encoders during the fine-tuning.

Performance was measured using mean Intersection over Union (mIoU) and macro F1 scores. A visual comparison for example patches is provided in Supplementary Figure 15.

Embedding Space Analysis The 2D visualizations shown in Figure 16 were generated by applying the Uniform Manifold Approximation and Projection (UMAP) [62] algorithm to the embeddings of every pixel within the Austrian study area. UMAP is a non-linear dimension-

Table 8. **Pre-training hyperparameters for the TESSERA v1.0 model.**

Description	Symbol	Value
Optimizer & Schedule		
Optimizer	-	AdamW
Batch Size (Per-GPU)	B_{local}	2,048
Batch Size (Global)	B_{global}	32,768 (16 GPUs $\times$ 2,048)
Epochs	-	1
Base Learning Rate	η	0.002
Weight Decay	λ_{wd}	1×10^{-6}
Learning Rate Schedule	-	Linear warmup (10%) followed by cosine decay
Gradient Clipping Norm	-	2.0
Loss Function		
Barlow Twins Lambda	λ_{BT}	5×10^{-3}
Mixup Lambda	λ_{mix}	1.0
Architecture & Data		
Sequence Length	L	40
Representation Dimension	D_{repr}	128
Projector Output Dimension	D_{proj}	16,384
Quantization Bits (QAT)	-	8

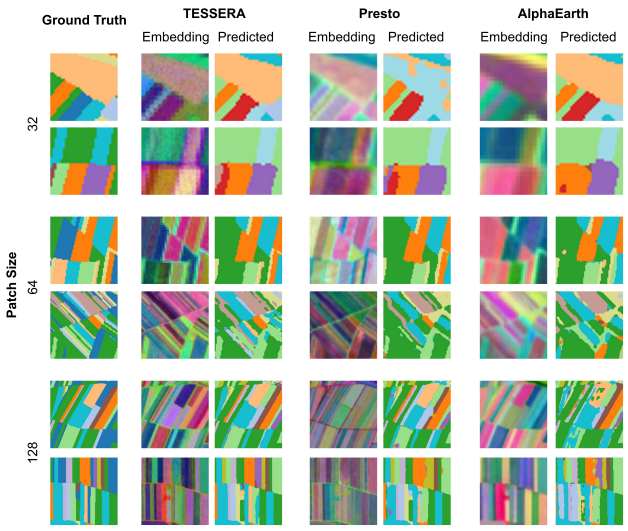

Figure 15. **TESSERA predictions most closely resemble the ground truth for patch-based semantic segmentation.** Each row corresponds to a different patch size (32x32, 64x64, 128x128). The first column shows the ground truth segmentation. For each model (TESSERA, Presto, and AlphaEarth), two columns are displayed: a PCA visualization of the input embeddings for the patch (labelled 'Embedding') and the final predicted segmentation map (labelled 'Predicted').

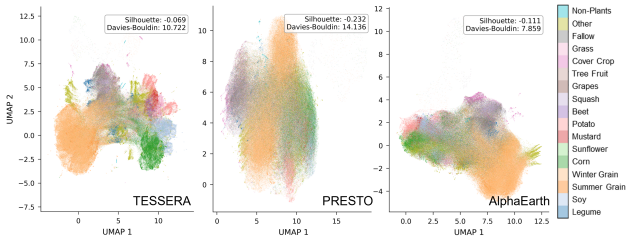

Figure 16. UMAP visualization of the embedding spaces for TESSERA, Presto, and AlphaEarth for 17 crop classes. TESSERA's embeddings exhibit clearer separation and more coherent clustering visually.

ality reduction technique based on Riemannian geometry and algebraic topology. It constructs a weighted k-nearest neighbour graph in the original high-dimensional space, then optimizes a low-dimensional projection to primarily preserve local distances and local structure. UMAP, like other non-linear techniques such as t-SNE and LargeVis, is designed to preserve local relationships within high-dimensional data; however, it is also able to retain more of the global structure, making it particularly effective for understanding both fine-scale and broader patterns in the embedding space. In this work, the entire embedding map (for example, a shape array $H \times W \times 128$, where H and W are the height and width of the region) was used as a direct input to UMAP for the TESSERA, Presto, and AlphaEarth embeddings.

To quantitatively measure the quality of the clustering in the original 128-dimensional space, we calculated the Silhouette score and the Davies-Bouldin Index.

- The **Silhouette score**, $s(i)$, for a single data point i measures how similar it is to its own cluster compared to other clusters. It is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{2}$$

where $a(i)$ is the mean distance between i and all other points in the same cluster, and $b(i)$ is the mean distance from i to all points in the nearest neighbouring cluster.

Table 9. **Ablation on the Austrian Crop dataset: Seasonal Data Inclusion.** We report the **Validation F1** score (higher ↑ is better) for the crop classification task. The columns for Spring to Winter use ✓ to indicate inclusion and ✗ to indicate exclusion. Parentheses show change vs. the Full Model.

Configuration	Spring	Summer	Autumn	Winter	F1 (↑)
Full Model (Baseline)	✓	✓	✓	✓	78.94
w/o Spring	✗	✓	✓	✓	78.62 (-0.32)
w/o Summer	✓	✗	✓	✓	78.10 (-0.84)
w/o Autumn	✓	✓	✗	✓	78.22 (-0.72)
w/o Winter	✓	✓	✓	✗	**79.83** (+0.89)

The score ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

- The **Davies-Bouldin Index** (DBI) evaluates clustering quality by computing the ratio of within-cluster scatter to between-cluster separation. For a set of k clusters, it is defined as:

$$\text{DBI} = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right) \tag{3}$$

where σ_i is the average distance of all points in cluster i to their centroid c_i, and $d(c_i, c_j)$ is the distance between the centroids of clusters i and j. Lower DBI values indicate better clustering, with a score of 0 representing the ideal case where the clusters are compact and well separated.

Impact of Seasonal Windows We analyze the impact of using different temporal windows on the model's performance for the downstream task of crop classification on the Austrian dataset. As shown in Table 9, we conduct an ablation by systematically excluding one of the four meteorological seasons (Spring, Summer, Autumn, Winter) from the input time series. The results reveal that, for this specific regional dataset, excluding time steps from the winter season improves the classification F1 score from 78.94% (all seasons included) to $\mathbf{79.83}\%$. This suggests that the dormant winter period, which is typically characterized by low vegetation phenological activity and is prone to cloud cover, snow, or reduced solar zenith angle effects, introduces a substantial amount of non-discriminative noise. By excluding the winter data, the model can focus its attention and resources on the most informative growing seasons, which allows for the learning of a more concise and effective temporal representation for crop type identification.

A.10. Downstream Task: Canopy height estimation

Study Areas and Ground Truth Datasets Canopy height prediction was evaluated in a 5×6 km area in the Danum Valley, Borneo. This region is covered in tropical rainforest dominated by species in the Dipterocarpaceae family, which hosts some of the tallest trees in the tropics [68, 79]. Ground truth data was derived from airborne laser scanning (ALS) performed in February 2020 [18]. ALS data provides detailed 3D point clouds of vegetation and terrain structure, which were processed using the LSPIKEFREE algorithm [30] to estimate top-of-canopy height at 1 m resolution. These canopy height rasters were then downsampled using mean pooling to 10 m to match the resolution of the TESSERA representations.

Modelling methods To predict the canopy height from the different embeddings, we used a standard U-Net architecture [75]. It takes as input a 32 × 32 pixel patch (covering a 320 × 320 m area) of representations. For 128-dimensional embeddings (as in TESSERA and Presto), this results in a 32 × 32 × 128 data cube. The network outputs a single-band 32 × 32 raster of predicted canopy height values. The proposed U-Net contains approximately 30 million parameters and was trained for 200 epochs using a batch size of four patches. At the end of each epoch, the model was evaluated using validation data and the model with the best performance on the validation set was retained for the final evaluation. In practice, training usually converged within the first 20 epochs, with rare boosts in performance later in training. Model training was done in Python using PyTorch.

To ensure spatial generalization and avoid overfitting local patterns, the reference data were divided into training, validation, and test sets that are spatially disjoint. Each study region was divided into four spatial folds based on cardinal directions (bottom, top, left, right), each fold covering 50% of the area. For each fold, the remaining half of the region was used for training and validation. To account for inherent model randomness, we trained three independent models per fold, resulting in 12 trained models for each foundation model. This strategy captures variability due to both spatial heterogeneity and training randomness. The final performance of the model was evaluated on the test data held using the following metrics: coefficient of determination (R^2), root mean square error (RMSE), and mean bias.

Baselines TESSERA, Presto [91], and AlphaEarth [13] embeddings were compared by training identical models using each representation as input, with all other settings held constant. Each model was trained for 200 epochs on the same data split, and the version achieving the best validation performance was selected for final evaluation on the held-out test set.

To benchmark performance against classical remote sensing approaches applied at the global scale, we compared the predictions against a global canopy height product: Tolan et al. [2024][89] (Meta). These maps were downloaded at 10 m resolution from Google Earth Engine and compared directly to the canopy height derived from

ALS. Although there are some temporal mismatches between ALS acquisition (2020) and global products, we assume relative stability of canopy height in the short term in this region, which is reasonable given that the area was not logged or burned.

Finally, we trained the U-Net model using Sentinel-2 input as a baseline. This approach retained the same model architecture, patch size, and evaluation setup, but replaced embeddings with cloud-free annual median composites of Sentinel-2 imagery. Cloud masking and compositing were performed in GEE, resulting in 12 channel input images, with one channel for each Sentinel-2 spectral band. Although different Sentinel-2 bands have different spatial resolutions, all were resampled to 10 m to match the FM pipelines.

A.11. Downstream Task: Above-ground biomass estimation

Dataset For the AGB regression task, we used the ground truth data from the BioMassters dataset [65]. This dataset covers nearly 11,500 forest patches in Finland, with each patch representing a 2,560×2,560 m area at 10 meter pixel resolution. Above-Ground Biomass (AGB) values were determined using open forest airborne LiDAR data and an extensive network of field plots as reference measurements by the Finnish Forest Centre. Patch coordinates and collection years–provided by the authors of Ref. [65] upon request–were used to retrieve satellite imagery to generate TESSERA and Google AlphaEarth Satellite Embeddings (AlphaEarth) [13].

UNet model architecture To obtain AGB estimates, we trained a convolutional neural network based on the UNet architecture [75], using TESSERA or AlphaEarth embeddings [13] as input. The model received 256×256×128 and 256×256×64 input patches for TESSERA and AlphaEarth, respectively, where the dimension denotes the number of channels.

The network followed a standard UNet structure with two encoder and decoder stages. The encoder comprised two convolutional blocks with 256 and 512 channels, each consisting of two 3×3 convolutions followed by batch normalization, ReLU activation, and 2×2 max pooling. A 1024-channel bottleneck block was used prior to upsampling. The decoder mirrored the encoder, using transposed convolutions for upsampling and skip connections from the encoder, followed by double-convolution blocks. A final 1×1 convolution produced the single-channel regression output. The model had approximately 30 million trainable parameters. The loss function was defined as the root mean squared error (RMSE) averaged over individual patches.

Following the evaluation protocol in Ref. [65], RMSE was computed on a test set of 2773 patches and reported as the average RMSE per patch. For limited label testing, a subset of labels was randomly chosen from the remaining data and randomly divided into training and validation sets (80:20). The UNet model was trained on the embeddings for 80 epochs, with the model achieving the lowest loss on the validation set selected as the final model. Figure 5 shows the average RMSE calculated over five random splits, with bars indicating the minimum–maximum range when it exceeds the size of the dot. Training was performed with Adam Optimizer (learning rate = 1e-4, batch size = 4). Task-specific training on the entire dataset (80% training / 20% validation) using a single NVIDIA H200 GPU and 8 CPU cores took approximately 5 hours for TESSERA embeddings and 4 hours for the AlphaEarth embeddings.

Baselines In addition to AlphaEarth, we compared the performance of TESSERA to two other baselines: the winner of the BioMassters competition [65] and the SpectralGPT foundation model [36] combined with a UPerNet [111].

For SpectralGPT, we used the implementation provided in Ref. [60], with the BioMassters satellite data as input, a lightweight spatial-temporal encoder L-TAE [76] for temporal aggregation, and a UPerNet as the regression head. This setup was found in [60] to outperform other tested foundation models on the BioMassters dataset. To ensure consistency in the limited label setting, we replaced the default sampling strategy with the same random splits used to evaluate TESSERA and normalized the inputs with their mean and standard deviation. Although we initially experimented with using per patch RMSE as the loss function to align with our evaluation metric, we ultimately reverted to the MSE loss [60] because it resulted in slightly better performance. Obtaining the results for the full dataset using a single NVIDIA H200 GPU took approximately 18 hours.

For evaluating the supervised baseline, we downloaded the pre-trained weights of the BioMassters winning model and ran inference on the test set, applying a post-processing step to clip predictions to a non-negative range. Although we attempted to retrain the model from scratch using two H200 GPUs and a different CUDA version than in the original setup, this configuration led to inferior performance compared to the original pre-trained model. According to the BioMassters codebase, the original model was trained on two A100 40GB GPUs over the course of eight days, highlighting the greater time efficiency of TESSERA, which required only five hours.

A.12. Interactive Habitat Mapping Tool

Here we demonstrate our interactive classification tool, a web-based utility designed for rapid human-in-the-loop habitat mapping using pre-computed model embeddings, as seen in Figure 18. The tool allows a user to first define a

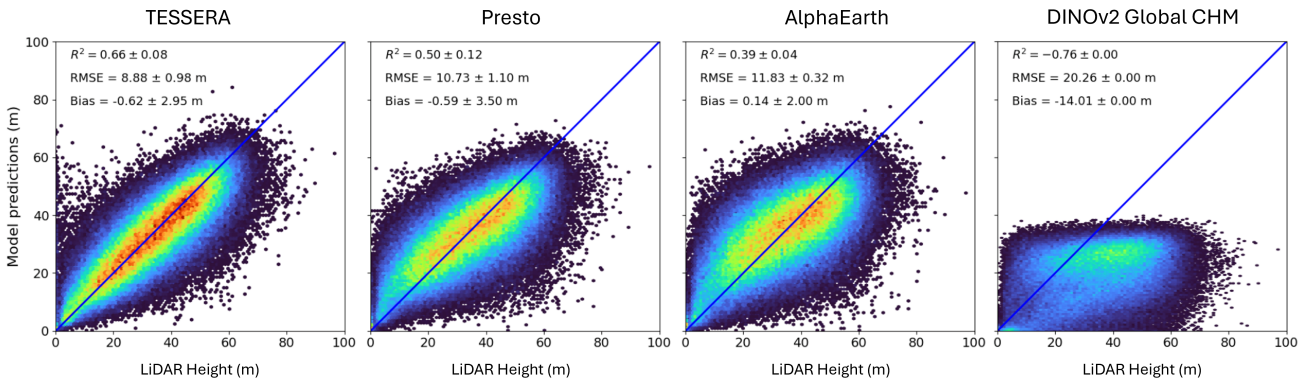

Figure 17. **Comparison of canopy height estimation performance across four models.** Predicted canopy height (y-axis) is compared against airborne LiDAR reference measurements (x-axis) in Danum Valley, Borneo. The figure shows scatter-density plots for four model variants trained using different input representations: **TESSERA**, **Presto**, **AlphaEarth**, and the **DINOv2 Global CHM** product. All models were evaluated using the same LiDAR reference dataset for consistency. TESSERA achieves the highest coefficient of determination and lowest RMSE, demonstrating superior performance in structurally complex tropical canopies. In contrast, the DINOv2 Global CHM product exhibits strong underestimation, large negative bias, and a near-zero or negative correlation, reflecting substantial saturation at lower canopy heights. A 1:1 reference line (blue) is included in each subplot for comparison.

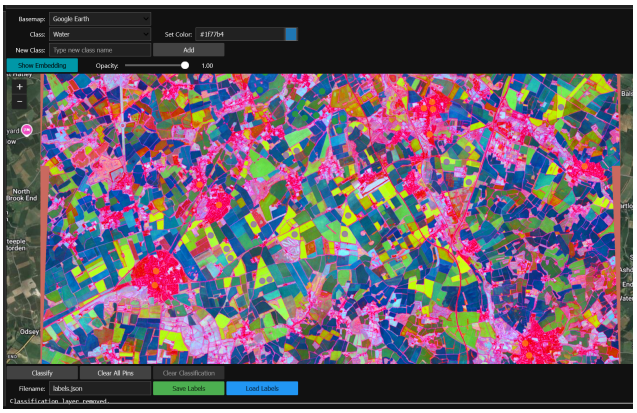

Figure 18. **We provide an interactive tool to allow non-specialist users to map TESSERA embeddings to user-defined classes.** We show a PCA-derived false colour map overlaid on an RGB aerial image of South Cambridge.

region of interest, for which it fetches and mosaics the relevant high-dimensional embedding tiles. To aid visual interpretation, it performs Principal Component Analysis on the embeddings and displays the first three components as a false-colour RGB image overlay on a satellite basemap.

The core of the tool is the interactive labelling workflow. A user can define custom habitat classes and provide training data by clicking on the map to place labelled points. These points and their corresponding 128-dimensional embedding vectors are used to train a k-Nearest Neighbors (kNN) classifier. The trained model then predicts a class for every pixel in the region of interest, producing a map layer that overlays a coregistered aerial view. This process allows for iterative refinement, where a user can add more points to correct misclassifications and retrain the model for immediate feedback, demonstrating an accessible method for quickly creating habitat maps without requiring deep machine learning expertise. We also provide a batch processing tool for larger regions of interest. The tools and their open source code are freely available in the github repository.

A.13. Pseudo Code

This section provides the pseudocode for mixup regularization and global shuffling.

Algorithm 1 Mixup Consistency Regularization for Multimodal Barlow Twins

```
def MixupBT(s2_aug1, s2_aug2, s1_aug1, s1_aug2, model, barlow_lambda, lambd_mix):
  # s2_* and s1_* are batches of augmented inputs
  # model returns projection z and representation repr
  # standard Barlow Twins branch
  z1, repr1 = model(s2_aug1, s1_aug1)
  z2, repr2 = model(s2_aug2, s1_aug2)
  loss_bt, bar_bt, off_bt = BarlowTwins(z1, z2)
  # mixup branch
  B = s2_aug1.shape[0]
  idx = randperm(B)
  alpha = Beta(beta_alpha, beta_beta).sample()
  # mix S2 and S1 views
  y_m_s2 = alpha * s2_aug1 + (1 - alpha) * s2_aug2[idx]
  y_m_s1 = alpha * s1_aug1 + (1 - alpha) * s1_aug2[idx]
  z_m, repr_m = model(y_m_s2, y_m_s1)
  # cross-correlation of mixed and original views
  cc_m_a = Corr(z_m, z1)
  cc_m_b = Corr(z_m, z2)
  cc_11 = Corr(z1, z1)
  cc_2idx1 = Corr(z2[idx], z1)
  cc_12 = Corr(z1, z2)
  cc_2idx2 = Corr(z2[idx], z2)
  # mixup targets and consistency loss
  cc_m_a_gt = alpha * cc_11 + (1 - alpha) * cc_2idx1
  cc_m_b_gt = alpha * cc_12 + (1 - alpha) * cc_2idx2
  diff_a = (cc_m_a - cc_m_a_gt).pow(2).sum()
  diff_b = (cc_m_b - cc_m_b_gt).pow(2).sum()
  loss_mix = lambd_mix * barlow_lambda * (diff_a + diff_b)
  # final SSL loss with mixup consistency
  loss = loss_bt + loss_mix
  return loss
```

Algorithm 2 Global Shuffling and Multimodal Data Construction

```
def GlobalShuffling(tile_roots, batch_size, T):
  # tile_roots:  list of paths to MGRS tile directories
  # batch_size:  number of tiles to process in memory simultaneously
  # T: number of time steps for sparse sampling
  tiles = shuffle(tile_roots)  # Randomize tile processing order
  num_batches = len(tiles) // batch_size
  for b in range(num_batches):
     current_tiles = tiles[b*batch_size : (b+1)*batch_size]
     pixel_buffer_aug1 = []
     pixel_buffer_aug2 = []
     # Parallel processing of tiles to extract valid pixels
     parfor tile in current_tiles:
        s2, s1, masks = load_data(tile)
        valid_locs = filter_valid_pixels(masks, s1, s2)
        if len(valid_locs) == 0:  continue
        # Sparse random temporal sampling per pixel
        for (y, x) in valid_locs:
           idx = get_valid_time_indices(masks, y, x)
           v1 = Sample(s2, s1, idx, T) # View 1
           v2 = Sample(s2, s1, idx, T) # View 2
           pixel_buffer_aug1.append(v1)
           pixel_buffer_aug2.append(v2)
     # Aggregate pixels from spatially disjoint tiles
     stack_aug1 = Stack(pixel_buffer_aug1) # Shape:  (N_total, T, C)
     stack_aug2 = Stack(pixel_buffer_aug2)
     # Global Shuffling:  Decorrelate spatial neighborhoods
     N_total = stack_aug1.shape[0]
     perm_idx = randperm(N_total)
     shuffled_aug1 = stack_aug1[perm_idx]
     shuffled_aug2 = stack_aug2[perm_idx]
     # Save as chunks for pre-training dataloader
     save_chunks(shuffled_aug1, shuffled_aug2, out_dir)
```

Table 10. **TESSERA provides unprecedented ease of use, scale, and accuracy.** Unlike other remote sensing foundation models, TESSERA provides analysis-ready and gap-free outputs (R), is open (O), and provides global (G) annual (A) coverage at 10 m resolution (10 m). It uses spectral-temporal features (ST) at pixel level (P), is invariant to clouds (CI), provides a compressed representation (CR), is oriented to land features (LO), and includes both multispectral and radar information (MS). Codes: **Y**= Yes, **N**= No, ? = Unknown, **P**= Potentially true, .. = N/A.

Foundation Model	R	O	G	A	10 m	ST	P	CI	CR	LO	MS
TESSERA (2025)	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
GeoKR (2021) [46]	**N**	?	**N**	**N**	**Y**	**N**	**Y**	**N**	**N**	**Y**	**N**
CVPRW (2021) [82]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	..	**Y**	**N**
GASSL (2021) [6]	**N**	..	**N**	**N**	**Y**	**N**	**Y**	**N**	..	**Y**	**N**
SeCo (2021) [61]	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**N**	?	**Y**	**N**
MOSAIKS (2021) [74]	**N**	**Y**	**Y**	**N**	**N**	**N**	**N**	**N**	**Y**	**Y**	?
DINO-MM (2022) [101]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	..	**Y**	**Y**
SatMAE (2022) [17]	**N**	..	**N**	**N**	**Y**	**Y**	**N**	**P**	..	**P**	**N**
RS-BYOL (2022) [41]	**N**	..	..	**N**	**Y**	**N**	**Y**	**Y**	..	**N**	**Y**
GeCo (2022) [47]	**N**	..	..	**N**	..	**N**	**N**	**N**	..	**P**	**N**
RingMo (2022) [83]	**N**	..	..	**N**	**Y**	**N**	**N**	**N**	?	**Y**	**N**
RVSA (2022) [96]	**N**	..	..	**N**	..	**N**	**N**	**N**	..	**Y**	**N**
RSP (2022) [95]	**N**	..	..	**N**	**P**	**N**	**N**	**P**	**Y**	**Y**	**N**
MATTER (2022) [2]	**N**	..	..	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**
CSPT (2022) [120]	**N**	..	..	**N**	?	**N**	**N**	**N**	**Y**	**Y**	**N**
CVPRW (2022) [77]	**N**	..	..	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**
BFM (2023) [14]	**N**	..	..	**N**	?	**N**	**N**	**N**	?	**Y**	**N**
TOV (2023) [87]	?	..	**N**	**N**	**Y**	**N**	?	**N**	?	**Y**	**N**
CMID (2023) [64]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	?	**Y**	**N**
RingMo-Sense (2023) [115]	**N**	..	**N**	**N**	**Y**	**Y**	**N**	**N**	?	**Y**	**Y**
lal-SimCLR (2023) [69]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**
CACo (2023) [59]	**N**	..	**N**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**
SatLas (2023) [8]	**N**	..	**Y**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**
GFM (2023) [63]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
Scale-MAE (2023) [73]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
DINO-MC (2023) [108]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
CROMA (2023) [31]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**Y**
Cross-Scale MAE (2023) [86]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
DeCUR (2023) [104]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**Y**	**N**	**Y**	**Y**
Presto (2023) [91]	**N**	..	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
CtxMIM (2023) [119]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
FG-MAE (2023) [102]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
Prithvi (2023) [78]	**N**	..	**N**	**N**	**N**	**Y**	**N**	**Y**	**Y**	**Y**	**N**
RingMo-Lite (2023) [103]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
IGARSS (2023) [29]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
EarthPTi (2023) [80]	**N**	..	**N**	**N**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**N**
USat (2023) [40]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
AIEarth (2023) [114]	**P**	?	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
Clay (2023) [16]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
Hydro (2023) [19]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**
U-Barn (2023) [26]	**N**	..	**N**	**Y**	**Y**	**Y**	**N**	**Y**	**N**	**Y**	**N**
GeRSP (2023) [38]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
SwiMDiff (2023) [88]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
OFA-Net (2023) [113]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**Y**
SML-FR (2023) [23]	**P**	**P**	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**

Continued on next page

Foundation Model	R	O	G	A	10 m	ST	P	CI	CR	LO	MS
Spectral-GP (2024) [36]	**N**	**N**	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
S2MAE (2024) [49]	**N**	**N**	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
SatMAE++ (2024) [67]	**N**	**N**	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
msGFM (2024) [34]	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
SkySense (2024) [33]	**N**	**N**	**N**	**N**	**Y**	**Y**	**N**	**N**	**N**	**Y**	**Y**
MTP (2024) [97]	**N**	..	**Y**	**N**	**Y**	**N**	**Y**	**N**	**N**	**Y**	**N**
DOFA (2024) [112]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**
MMEarth (2024) [66]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**
LeMeViT (2024) [43]	**N**	..	**N**	**N**	?	**N**	**N**	**N**	**Y**	**N**	**N**
SoftCon (2024) [105]	**N**	..	**Y**	**N**	**Y**	**N**	**Y**	**N**	**N**	**Y**	**N**
RS-DFM (2024) [107]	**N**	..	**N**	**N**	**Y**	**N**	**Y**	**N**	**N**	**Y**	**N**
A2MAE (2024) [118]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**N**
HyperSIGM (2024) [98]	**N**	..	**Y**	**N**	**Y**	**Y**	**Y**	**N**	**Y**	**Y**	**N**
SelectiveMAE (2024) [99]	**N**	..	**Y**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**N**
OmniSat (2024) [4]	**N**	..	**N**	**Y**	**Y**	**Y**	**N**	?	**Y**	**Y**	**Y**
MM-VSE (2024) [72]	**N**	..	**Y**	**Y**	**Y**	**Y**	**N**	?	**Y**	**Y**	**Y**
MA3E (2024) [52]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**N**
SpectralEarth (2024) [12]	**N**	..	**Y**	**Y**	**N**	?	**Y**	**N**	**Y**	**Y**	**N**
SenPa-MAE (2024) [70]	**N**	..	**Y**	**Y**	**Y**	**Y**	**Y**	**N**	**Y**	**Y**	**N**
RingMo-Ae (2024) [21]	**N**	..	**N**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**N**
SAR-JEPA (2024) [48]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**Y**	**N**	**N**	**N**
PIS (2024) [3]	**N**	?	**Y**	**Y**	**Y**	..	..	..	..	..	..
OReole-FM (2024) [22]	**N**	..	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**N**
PIEVIT (2024) [56]	**N**	..	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**N**
SatVisionTOA (2024) [81]	**N**	..	**Y**	**Y**	**N**	**Y**	**Y**	**N**	**Y**	**Y**	**N**
RS-vHeat (2024) [37]	**N**	..	?	?	?	?	**Y**	?	**Y**	?	**Y**
Prithvi-EO-2.0 (2024) [84]	**N**	..	**Y**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**Y**	**N**
AnySat (2024) [5]	**N**	..	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**Y**
WildSAT (2024) [20]	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**	**Y**	**N**
SeaMo (2024) [51]	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**N**
FoMo (2025) [11]	**Y**	**Y**	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**Y**	**Y**
SatMamba (2025) [25]	**Y**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**	**N**
Galileo (2025) [92]	**P**	**P**	**Y**	?	**Y**	**N**	?	?	**Y**	**Y**	**Y**
SatDiFuser (2025)[42]	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
RoMA (2025) [100]	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**
Panopticon (2025) [94]	**N**	**Y**	**N**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**Y**
FedSense (2025) [85]	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**Y**	**N**
Copernicus-FM (2025) [106]	**N**	..	**Y**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**Y**
HyperFree (2025) [45]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**N**
DynamicVis (2025) [15]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**Y**
FlexiMo (2025) [50]	**N**	..	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**Y**	**Y**
Google Satellite (2025)[13]	**Y**	**N**	**Y**	**Y**	**Y**	**Y**	**N**	**Y**	**Y**	**Y**	**Y**
SkySense++ (2025) [109]	**N**	**Y**	**Y**	**P**	**P**	**Y**	**N**	**P**	**Y**	**Y**	**Y**
RingMoE (2025) [9]	**N**	..	**N**	**N**	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**Y**