OlmoEarth v1.Embeddings

Team OlmoEarth*

Gabriel Tseng **Favyen Bastani** **Patrick Alan Johnson** **Yawen Zhang** **Piper Wolters**

Ando Shah **Hadrien Sablon** **Joseph Redmon** **Heather DeHaven** **Rachel Ratner**

Sasha Levy **Jordan Steward** **Katrina Midgley** **Andrew Howe** **Hunter Pitelka** **Ted Schmitt**

Allen Institute for AI

*OlmoEarth is a team effort.

Embeddings: olmoearth.allenai.org/embeddings

Training Code: olmoearth_pretrain

OlmoEarth Pre-trained Model: OlmoEarth-v1_3-Base

OlmoEarth Pre-training Dataset: olmoearth_pretrain_dataset

Contact: olmoearth@allenai.org

Abstract

We make some OlmoEarth embeddings. To do this, we train a new model: OlmoEarth v1.3. OlmoEarth v1.3 is designed to condense multimodal, multitemporal remote sensing data into a single digestible 64 or 128-dimensional vector. OlmoEarth v1.3 produces state-of-the-art embeddings. In addition, OlmoEarth v1.3 is a strong encoder backbone – competitive with OlmoEarth v1 and v1.2 – while requiring 18% and 53% of the GPU hours to train, respectively. We use OlmoEarth v1.3 to compute global, annualized 10m embeddings, which we release under the OlmoEarth artifact license.

1 Introduction

Embeddings are an extremely convenient way to represent Earth Observations. When making large scale maps, inference costs dominate: Section 4 of the OlmoEarth v1.2 report [27] notes that when making maps at scale with OlmoEarth v1.2, 98% of the cost is inference through the model and 2% of the cost is data export & preprocessing. Embeddings bear the inference cost once but can be reused for many downstream tasks. In addition, embeddings enable use cases not possible when finetuning models, such as aggregating information over large scales [19].

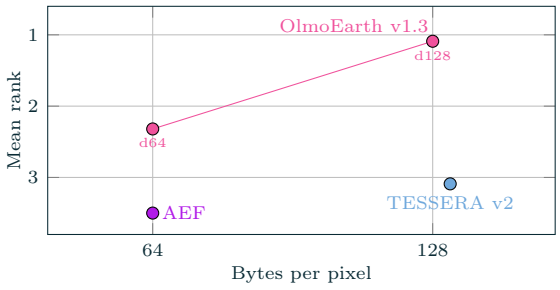


Figure 1 Storage cost per pixel against mean rank (the smaller the better) over 11 evaluations (Section 3.1)[1], all four products ranked together. TESSERA v2 stores 128 int8 values plus a float scale.

Previous generations of OlmoEarth models (OlmoEarth v1 [12] and v1.2 [27]) have principally been deployed via finetuning within OlmoEarth Studio (https://olmoearth.allenai.org/platform). This tech report introduces OlmoEarth v1.3, a model which makes embedding a first class deployment target.

[1]For the AEF evaluations, we rank models using a single classifier. We use kNN-20 for consistency with Section 3.2.1.

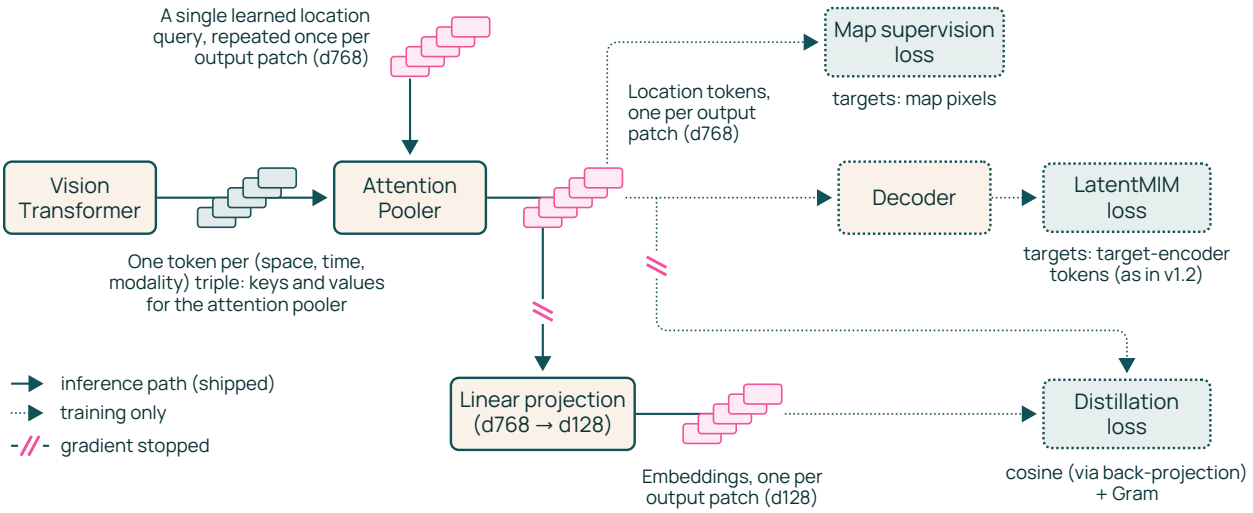


Figure 2 One OlmoEarth v1.3 training step. In a single forward pass, we compute the LatentMIM Lite and supervision losses (which are used to train the Attention Pooler and Vision Transformer) and the distillation losses (which are used to train the student). This makes distillation essentially free: no additional forward passes are necessary to compute the distillation losses.

Embeddings introduce three important considerations: (i) **aggregation**: combining tokens from multiple timesteps and modalities into a single representation per location, (ii) **efficiency** at high resolutions and (iii) **compaction**: decoupling the width of the backbone from the final embedding size.

OlmoEarth v1.3 yields state-of-the-art embeddings. When compared with the other two prominent 10m satellite embedding products – AlphaEarth Foundations [3] and TESSERA v2 (specifically TESSERA-v2-Large) [6] – OlmoEarth v1.3's 128-dimensional embeddings are best on 10/11 tasks[2], and our 64-dimensional embeddings beat both baselines on 7/11 tasks. In addition, OlmoEarth v1.3 is a strong finetuning model, maintaining OlmoEarth v1.2's quality despite requiring about half the GPU hours to train.

In Section 2 we describe OlmoEarth v1.3, focusing on how we address the aggregation, efficiency and compaction considerations necessary to make strong global-scale embeddings. In Section 3, we evaluate OlmoEarth v1.3's embeddings, and the backbone itself using the same kNN/LP and finetuning evals used in OlmoEarth v1.2. In Section 4, we describe our process for generating and serving global, multi-year embeddings from OlmoEarth v1.3. In Section 5, we discuss the environmental impact of pretraining OlmoEarth v1.3 and of deploying global, annualized embeddings.

2 OlmoEarth v1.3

The OlmoEarth models (v1 [12], v1.1, v1.2 [27] and v1.3) are encoder-decoder vision-transformer models trained via masked image modeling. The models are trained on multimodal, multitemporal remote sensing inputs. During training, part of the input is masked, and the model learns to reconstruct the masked part and other map targets based on the unmasked portion.

Previous generations of OlmoEarth models were principally deployed within OlmoEarth Studio via finetuning; v1.3 makes embeddings a first-class deployment target. Unlike finetuning, three considerations are introduced when producing embeddings: (i) **aggregation**, (ii) **efficiency** at high resolutions and (ii) **compaction**.

Below, we describe how we handle these considerations. These represent the changes between OlmoEarth v1.2 and v1.3. We finish by summarizing the overall OlmoEarth v1.3 architecture and training loop.

[2] As in Figures 1 and 7, we only consider the kNN-20 classifier for this summary.

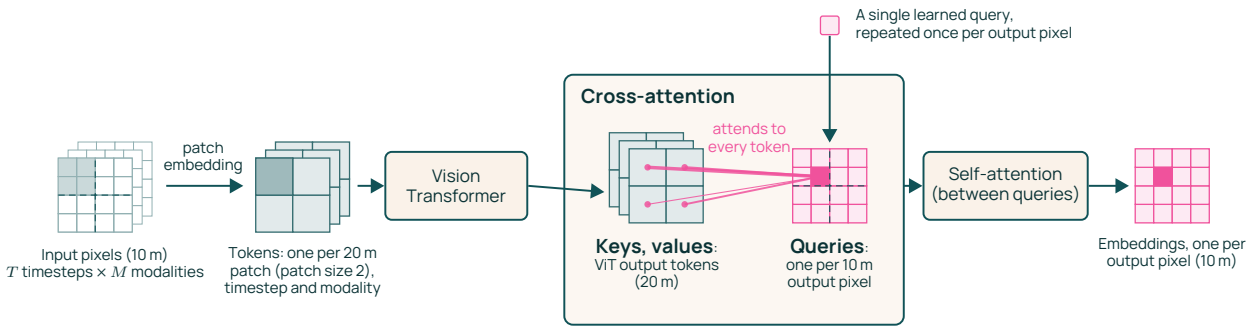


Figure 3 The Attention Pooler learns a query which cross-attends to the ViT's output tokens. We repeat this query at the desired output resolution to obtain embeddings at a finer granularity than the ViT's patch size; this decouple the model's output resolution from the number of tokens passing through the ViT (and therefore its compute cost).

2.1 Aggregation

The OlmoEarth models are based on vision transformers. Given an input of remote sensing data, we tokenize it before passing it through a number of attention blocks. This tokenization happens across three dimensions: space, time and modalities. Spatially, we define some patch size p and divide our inputs $H \times W$ into $p \times p$ patches. A token consists of one modality, one timestep and one $p \times p$ patch so that a T-timestep, 3 modality (Sentinel-1, Sentinel-2 and Landsat) input will have $T \times 3 \times \frac{H}{p} \times \frac{W}{p}$ tokens in total. When constructing embeddings, we only want a single token per spatial patch, so we must aggregate the modality and time dimensions to produce $\frac{H}{p} \times \frac{W}{p}$ tokens.

Naively averaging these embeddings can lead to detrimental results - for example, OlmoEarth v1.2 sees mixed results when combining modalities, even when finetuned (Table 2 of [27]). This makes intuitive sense; nothing during training tells the model that tokens will be averaged in this way. In addition, naive averaging is probably not optimal anyway; some tokens matter much more than others. For example, during cloudy months Sentinel-1 is likely to be more important.

We aggregate the encoder's output tokens using an Attention Pooler (a relatively common approach [13, 14, 17, 18]). Specifically, we learn a query and the "tokens to be aggregated" are the keys and values. In our case, the key / value tokens are the outputs of the vision transformer. This allows the vision transformer's output tokens to be aggregated using a data dependent aggregation function – the attention mechanism. Our Attention Pooler consists of two attention layers; a "read" cross-attention layer, and a latent self-attention layer (where the queries, keys and values are all the query tokens). During the reads, the keys and values are the final outputs of the Vision Transformer; we experimented with passing intermediate embeddings from the Vision Transformer too but this did not meaningfully change results.

We have two requirements for these query tokens: (i) locality – each query token must correspond to a specific spatial patch, and (ii) window invariance – windowing is arbitrary, so nothing should depend on a token's absolute position within the window. We achieve this with three mechanisms:

A single learned location query which is repeated to cover all spatial patches. This ensures window invariance, since every spatial patch shares the same query parameters. We refer to the location queries after they have passed through the Attention Pooler as "location tokens".

RoPE spatial encodings, applied during both the read and self-attention layers, give each query its spatial identity. Because RoPE is a *relative* positional encoding, it allows us to provide this spatial identity without imposing a dependency on a query token's absolute position within a window.

Direct supervision on location tokens. This direct supervision leverages our map modalities: we add a loss on each query token, training it to reconstruct the map pixels it covers spatially. Concretely, each location token covers a $p \times p$ spatial area in the input. For each map modality, we learn a linear projection (which we discard after pretraining) which maps each 768-dimensional output token to $P \times P \times n$ where P is our maximum location token patch size and n is the number of channels in that map. We then bilinearly resize to the token's footprint and apply an L1 loss for continuous maps (like SRTM) and a cross-entropy loss for

categorical maps (like WorldCover). The full supervision loss is a weighted sum of each map m's loss:

$$\mathcal{L}_{\text{sup}} = \sum_m w_m \mathcal{L}_m$$

For the continuous maps, a mean prediction incurs a loss of $\mathbb{E}|x - \mu| \leq \sigma$; under our normalization, $\sigma \approx 0.25$. For the categorical maps, a uniform prediction incurs a loss of $\ln C$, which is 2.4 for WorldCover's 11 classes and 4.9 for CDL's 133 classes. We therefore apply a weight $w_m = 1$ for the continuous maps and $w_m = 0.1$ for categorical maps to balance out these order-of-magnitude differences.

Direct supervision improves embedding performance; PASTIS validation mIoU increases from 51.3 without supervision to 58.4 with supervision (we describe how we run PASTIS in Section 3.1).

2.2 Efficiency at high resolutions

All OlmoEarth models use a flexible patch-embedding layer; this allows users to control the inference cost of their models (Figure 4). This patch size defines the resolution at which an input is tokenized; given a 10m per pixel input, a patch size of 2 means that each token represents a 20m × 20m spatial patch. In a standard ViT, the input resolution is equal to the output resolution; that is, at patch size 2 the ViT will also *output* tokens covering a 20m × 20m spatial patch.

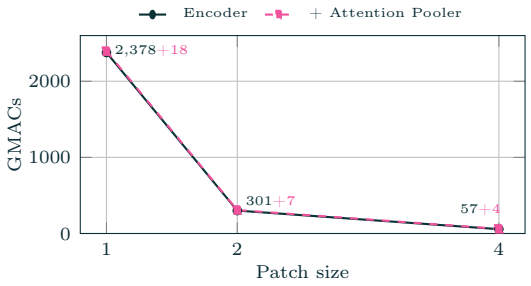


Figure 4 Inference cost of one 16 × 16 embedding window (12 timesteps, Sentinel-1, Sentinel-2 and Landsat) by patch size, with and without an Attention Pooler outputting at pixel-resolution. Labels give the encoder's GMACs and the Pooler's overhead.

During finetuning, this is not an issue; we can use a decoder head which upsamples these tokens to predictions at the desired resolution (typically 10m per pixel). However, when producing embeddings we don't have a decoder head - we'd like to decouple the encoder patch size (which dictates the inference compute cost) from the model's output resolution. The Attention Pooler provides a simple way to do this: we can repeat the learned queries for every desired output patch, instead of for every encoder spatial patch (Figure 3).

This allows us to produce 10m embeddings much more cheaply; Figure 4 shows the cost of adding an Attention Pooler to output 10m embeddings when running the ViT at different resolutions. The Attention Pooler adds a negligible cost (18 to 4 GMACs per window) while allowing significantly cheaper forward passes.

Decoupling the ViT patch size from Attention Pooler's output resolution allows us to efficiently produce 10m embeddings - running a ViT + Attention Pooler at encoder resolution 20m, output resolution 10m is **7.7× cheaper** than running a ViT at 10m resolution: 308 vs. 2,378 GMACs per window (for reference, TESSERA v2 costs around 700-1,200 GMACs for the same area, depending on imagery availability). Figure 4 shows the overhead of the Attention Pooler outputting at 10m resolution at different input patch sizes. The Attention Pooler introduces a negligible additional cost to the ViT.

We already sample encoder patch sizes p_e during training (Section 2.2 of [12]). We now additionally sample pooling patch sizes p_p such that $p_p \leq p_e$ - this ensures the pooler can output location tokens at different resolutions. We want the Attention Pooler's location tokens to retain fine-grained details. We use two mechanisms to encourage this; **direct supervision** on the output tokens, as described in Section 2.1, and **PixLatentMIM Lite**. This is a variation of our LatentMIM Lite loss (Section 2.4 of OlmoEarth v1, [12]). The targets (and therefore the decoder's mask tokens) always operate at a pixel resolution (patch size 1). To prevent memory issues from too many decode-tokens, we sample one pixel per masked patch to use as a target. This yields a small improvement over standard LatentMIM Lite (+0.5 mIoU on the PASTIS validation set).

2.3 Compaction

A ViT-Base has 768-dimensional encodings; we would like to decouple the final output size from the model's capacity, so that much more compact embeddings can be produced. We tried two ways to achieve this:

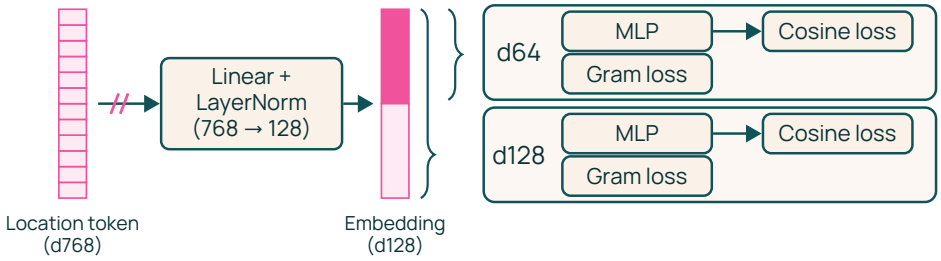


Figure 5 We use a linear layer to distill the 768-dimensional location tokens to 128-dimensional embeddings. We apply the distillation losses to the 128-dimensional embeddings and to their first 64 dimensions.

query compaction, where the Attention Pooler's queries are set at a desired dimensionality (e.g. 128), and **distillation**, where the entire backbone is kept at 768 dimensions before being shrunk via distillation.

Both of these approaches yielded similar performance on the 128d embeddings (distillation obtains an mIoU of 58.4 on the PASTIS validation set while token compaction obtains 57.5). Our final recipe uses distillation; we chose this because distillation offers two advantages over query compaction:

- **A strong teacher model**: Using 128 dimensional query tokens in the Attention Pooler forces a bottleneck through the Attention Pooler. When using distillation, the backbone is trained natively at d768. This ensures we have a strong teacher model, which also excels at finetuning.
- **Nested embeddings** via a Matryoshka learning [16], allowing users to select d128 or d64 embeddings based on their own performance vs. cost tradeoffs.

Our "distillation" consists of learning a single linear layer, which projects the 768 location tokens down to 128 dimensions, followed by a LayerNorm (Figure 5). To train this linear layer, we use two losses: a cosine similarity loss ($\mathcal{L}_{\text{cosine}}$) and a Gram loss ($\mathcal{L}_{\text{Gram}}$, following DINOv3 [22]). The cosine similarity loss uses a 2-layer MLP with hidden width 256 to project the student embeddings back to 768 dimensions, before measuring their cosine similarity with the location tokens. The Gram loss randomly samples 2,048 tokens from the batch and constructs two cosine-similarity matrices for these tokens: one using the 768 dimensional location tokens, and one using the 128 dimensional student embeddings. An MSE loss is then applied to compare the two matrices. Unlike DINOv3's Gram loss, this means similarities are also computed between tokens in different instances. Intuitively, the cosine loss encourages effective distillation at a token level, while the Gram loss encourages the student to preserve the overall geometry of the tokens by matching the teacher's pairwise similarities between tokens.

These distillation losses are **only used to train the linear projection** and the MLPs (which we discard after pretraining) – gradients from the distillation losses do not flow to the ViT backbone or the Attention Pooler. We use a Matryoshka loss for the distillation; this means that the distillation losses are applied twice, once to the first 64 dimensions of the student tokens and once to the full 128 dimensions (we learn one MLP per prefix for the cosine loss). This allows us to treat the first 64 dimensions of the student tokens as an even smaller set of embeddings.

We train our student jointly with the teacher (the ViT backbone and the Attention Pooler) rather than in a separate stage. Since the student only consists of a linear layer, this makes the distillation essentially free; a single forward pass allows us to compute the LatentMIM Lite and supervision losses for the teacher as well as distillation losses for the student. Gradients from the distillation losses are only used to train the linear distillation layer; they don't flow back to the Attention Pooler or the encoder.

2.4 Putting it together

OlmoEarth v1.3 follows the same encoder-decoder training recipe and architecture as OlmoEarth v1.2, but adds two components: the Attention Pooler (Section 2.1) and token distillation (Section 2.3). We use a ViT-Base for our Vision Transformer; this ViT-Base is identical to the OlmoEarth v1.2 Base model. The decoder inputs when training OlmoEarth v1.3 are the attention pooler outputs, not the raw outputs from the

Vision Transformer backbone. We provide a visual overview to OlmoEarth v1.3 in Figure 2.

All losses are simply summed during training:

$$\mathcal{L} = \mathcal{L}_{\text{LatentMIM}} + \mathcal{L}_{\text{sup}} + \mathcal{L}_{\text{cosine}} + \mathcal{L}_{\text{Gram}}$$

Where $\mathcal{L}_{\text{LatentMIM}}$ is the token-contrastive loss we used to reconstruct masked Sentinel-1, Sentinel-2 and Landsat tokens and the maps, $\mathcal{L}_{\text{sup}}$ is the supervision loss on maps, and $\mathcal{L}_{\text{cosine}}$ and $\mathcal{L}_{\text{Gram}}$ are cosine similarity and Gram loss used for distillation.

We use the same training data as OlmoEarth v1.2:

Observations	Maps	
Sentinel-1	WorldCereal [29]	OpenStreetMap [21]
Sentinel-2	WorldCover [33]	Cropland Data Layer [28]
Landsat-8	SRTM [20]	Canopy Height Map [24]

One of the main motivations for the "Instance Contrastive Loss" ([12], Section 2.4.2) is to pool tokens during pre-training so that when we pool tokens during deployment this is something the model has seen before. The Attention Pooler now aggregates tokens both during pre-training and deployment, allowing us to remove the instance contrastive loss without seeing a regression in performance. This speeds training up, since gradient updates now only require one forward pass per microbatch instead of two.

Aside from these changes, we train OlmoEarth v1.3 using the same recipe as OlmoEarth v1 and v1.2. This consists of using an AdamW optimizer with a batch size of 512 for 667,200 steps. We use a linear warmup of 8,000 steps and then apply a cosine annealing learning rate for the remainder of training. **OlmoEarth v1.3 cost** 53% **of the GPU hours required to train OlmoEarth v1.2, and** 18% **required to train OlmoEarth v1**.

3 Experiments & Results

3.1 Embedding evaluations

	Africa crop mask				Canada crops (coarse)				Canada crops (fine)				Descals oil palm				Ethiopia crops				GLanCE				LCMAP land use				US trees			
	AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3		AEF	TES.	v1.3	
Readout	64	128	128	64	64	128	128	64	64	128	128	64	64	128	128	64	64	128	128	64	64	128	128	64	64	128	128	64	64	128	128	64
Ridge, λ=0	88.8	88.1	91.9	**92.6**	70.3	70.7	**73.6**	73.3	60.6	64.5	**66.2**	63.5	77.4	69.8	**79.1**	79.0	68.5	68.1	63.6	**69.1**	67.7	68.7	**70.9**	69.5	85.8	85.2	**88.7**	88.1	21.1	20.9	**23.0**	21.2
kNN-5	87.6	88.3	**91.7**	91.6	71.2	71.0	**72.4**	71.5	61.3	62.6	**64.7**	64.0	71.2	69.0	**72.0**	71.6	66.4	**77.4**	70.8	69.2	68.0	68.0	**70.2**	70.0	83.4	83.7	86.3	**86.3**	18.3	16.1	**18.6**	18.3
kNN-20	86.9	87.1	**91.8**	91.6	70.9	70.6	**71.7**	70.9	61.9	63.3	**64.9**	63.9	72.2	70.3	**73.2**	72.8	64.3	**74.2**	67.0	66.2	68.8	68.8	**71.0**	70.8	84.2	84.2	**87.1**	87.0	21.9	19.5	**22.0**	21.6
Ridge, λ=1	89.1	89.7	**93.2**	93.0	70.1	71.5	**74.5**	73.3	60.1	64.5	**66.0**	63.1	76.4	71.5	**80.6**	78.4	66.2	**79.6**	73.8	70.3	67.7	68.7	**70.9**	69.5	85.7	85.2	**88.7**	88.1	21.1	20.9	**22.9**	21.3
Ridge, λ=10	86.8	90.1	**93.3**	92.9	68.1	71.5	**74.0**	72.3	57.6	64.1	**65.1**	62.3	72.9	72.4	**79.8**	77.0	60.2	**80.1**	67.1	64.9	67.4	68.6	**70.8**	69.4	85.2	85.2	**88.7**	88.1	20.8	20.8	**22.8**	21.2
LogReg	87.9	89.8	**93.0**	92.8	69.3	70.4	**74.4**	73.4	62.0	64.5	**66.8**	65.7	75.6	71.6	**81.3**	79.1	63.3	**79.7**	71.0	67.9	69.1	68.7	**71.0**	70.9	86.2	84.6	**88.4**	88.2	22.3	22.0	**23.9**	22.9
RF	88.5	89.6	**92.5**	92.0	71.0	71.5	**72.7**	71.9	62.7	64.7	**65.4**	64.7	74.3	71.7	**77.9**	77.0	68.8	**77.3**	71.3	70.1	69.2	69.5	**71.3**	71.0	84.4	84.1	**87.0**	86.8	**22.4**	20.3	21.9	21.5
MLP	89.6	90.1	**92.9**	92.5	71.2	71.7	**73.7**	73.2	62.2	63.1	**65.7**	64.5	76.6	72.7	**80.5**	78.8	71.9	**80.4**	75.9	74.0	68.3	67.0	**69.9**	69.2	85.4	84.4	**87.8**	87.6	**17.6**	15.3	17.2	17.5

Table 1 Balanced accuracy on the eight AEF balanced-trial datasets for every readout. Sub-columns are AEF (64 dims), TESSERA v2 ("TES.", 128 dims) and OlmoEarth v1.3 at 128 dims and its first 64 dims. The first block is the AEF protocol (ridge with λ=0, kNN with k=5, 20); the second block adds commonly used classifiers at library defaults. Every classifier is fit on the same class-balanced draws at AEF's Table-1 sizes and scored on the remainder; cells are means over repeated draws. We highlight OlmoEarth cells that beat the better of the two baselines.

We compute our embeddings using all three input modalities (Sentinel-1, Sentinel-2 and Landsat). We use 12 input timesteps (to obtain annualized embeddings), a 16 × 16 input window, an encoder patch size of 2 and an output patch size 1 (yielding embeddings at a 10m × 10m resolution). In this setting, **the Attention Pooler is essentially free in terms of compute cost**, costing 7GMACs per window (a 2% increase from 301GMACs per window without the Attention Pooler to 308GMACs per window with the Attention Pooler).

We compare our 128-dimensional embeddings with Tessera v2 Large [6] and Alpha Earth Foundations [3] (AEF). When running these evaluations, we aim to reflect the deployed results as closely as possible. For AEF, we retrieved the embeddings using Google Earth Engine [9]. For Tessera v2, we followed the inference

instructions in the official github repository[3]. For OlmoEarth, we compute embeddings using Sentinel-1, Sentinel-2 and Landsat as inputs with the same processing as pretraining (monthly least-cloudy imagery). **In all cases, we only consider annualized embeddings** since these reflect the most accessible products for all three models. Each embedding is aligned to its labels' calendar year. **In all cases, we quantize the embedding** using the deployed quantization method.

We evaluate our embeddings following the protocol from AEF. Specifically, for each of the AEF evals (Africa crop mask, Canada Crops {coarse, fine}, Descals Oil Palm, Ethiopia Crops, GLanCE, LCMAP land use, US trees) we sampled a balanced training set. For each classifier we evaluate, we train a classifier on this training set and evaluate on all non-sampled points. We do multiple draws per dataset, and report mean accuracy.

		PASTIS	PLANTEUR	
Model	Dim.	Random	Random	LOIO
AEF	64	39.1	27.7	15.8
TESSERA v2	128	58.3	34.2	19.1
OlmoEarth v1.3 distilled	128	**58.6**	**36.7**	**24.9**
OlmoEarth v1.3 distilled	64	53.4	35.2	21.5

Table 2 Crop-type segmentation results, measured via mIoU, on the PASTIS and PLANTEUR datasets. We also measure PLANTEUR's "leave-one-island-out" (LOIO) mIoU performance, which evaluates the embeddings' geographic generalization capabilities.

We additionally evaluate the embeddings on two crop type classification datasets: PASTIS [8] and PLANTEUR [32]. Like PASTIS, PLANTEUR is based on the Registre parcellaire graphique [5] but uses labels from Overseas France (Martinique, Réunion, French Guiana, Guadeloupe, Mayotte). PLANTEUR's geographic diversity allows us to asses test performance on one island when classifiers are trained on the other islands ("leave-one-island-out", LOIO). This allows us to assess the geographic generalization of the embeddings. Embeddings from these larger datasets don't fit in memory, so unlike the AEF evaluations we use a linear probe to assess embedding performance (with the same learning rate grid as Section 3.3 of [12]).

AEF and Tessera v2 are not the same size as OlmoEarth (128 dimensions, 128 bytes at int8). AEF is smaller (64 dimensions, 64 bytes at int8) while Tessera v2 is marginally bigger (128 dimensions but with a magnitude float as well, yielding 132 bytes at int8).

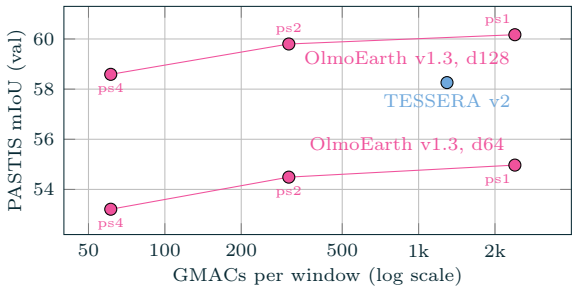


Figure 6 PASTIS validation mIoU against inference cost per 16×16×12 S1+S2+L8 window, for the 128-dimensional embeddings and their first 64 dimensions (d64). In all cases, the model's output patch size is 1 (10m). AEF scores 39.1.

These evaluated embeddings are not bit-identical to the released embeddings. Unlike the released embeddings, we computed these embeddings by tiling the ViT - the released embeddings use Searchlight for artifact free inference (Section 4.2). We measured the difference in performance on PASTIS with and without Searchlight inference, and observed a negligible difference (PASTIS val mIoU was 60.4 with Searchlight compared to 59.8 without).

3.1.1 Comparison with AEF and Tessera v2

Across our embedding evaluations, the OlmoEarth v1.3 embeddings are the best performing. We follow AEF in using kNN and a ridge regression with $\lambda = 0$ to evaluate the embeddings; this minimizes the classifiers' hyperparameters, ensuring methods aren't penalized for sub-optimal hyperparameters (Section S4.1 from [3]). Under this protocol, our d128 embeddings outperform AEF and TESSERA v2 on 7/8 AEF datasets. Our d64 embeddings outperform both on 5 to 7 of 8. The one exception is Ethiopia, where TESSERA v2 consistently outpeforms OlmoEarth v1.3. On crop-type-segmentation, our d128 embeddings slightly outperform TESSERA v2 on PASTIS (58.6 vs. 58.3 mIoU) and are best on PLANTEUR, while our d64 embeddings outperform AEF on PASTIS and both AEF and TESSERA v2 on PLANTEUR.

When deploying models practitioners are likely to experiment with a range of different classifiers. To reflect this, we run the AEF evaluations with a range of commonly used classifiers, using library defaults (Table

[3] https://github.com/ucam-eo/tessera/tree/master/tessera_infer_v2

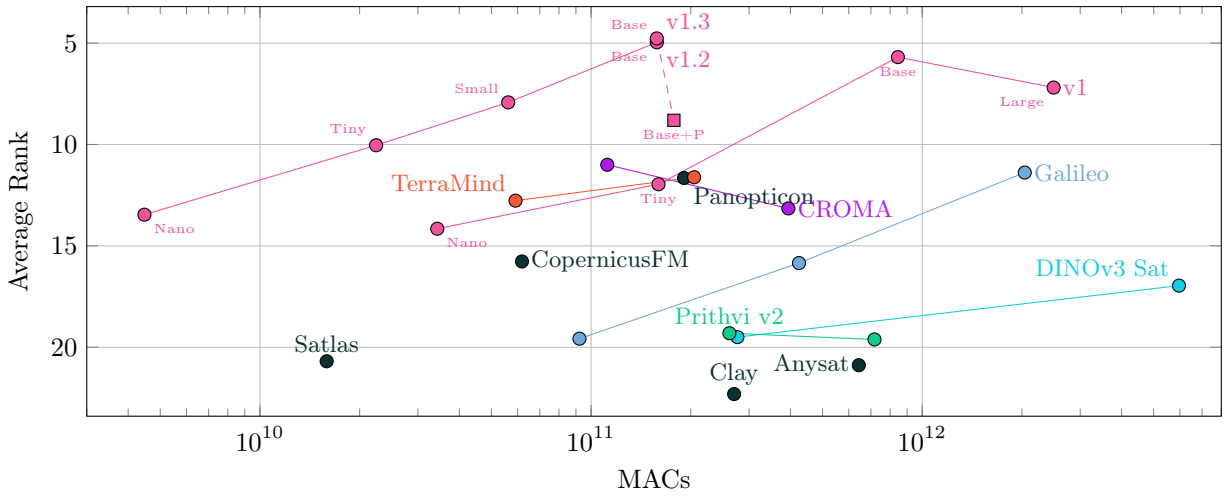


Figure 7 Performance vs. computational efficiency, averaged across 13 kNN/LP tasks (Sentinel-2 tasks which all models can support). The chart shows average Multiply-Accumulate operations (MACs) to encode one example across all tasks (input size varies by task) - each line connects model sizes within a family. We measure OlmoEarth v1.3 performance without the Attention Pooler and with (+P, square). Without the Attention Pooler, v1.3 has the best average rank of all models, at the same cost as v1.2 Base; on these 13 tasks, the Attention Pooler adds 13% to the MACs. Full results in Table 6.

1). Different classifiers do not significantly change the overall ranks of the models: our d128 embeddings remain best on 7/8 datasets under every linear classifier and 6/8 under random forests and the MLP. For OlmoEarth and TESSERA, adding some regularization to linear models improves performance - the Ridge ($\lambda = 1$) and Logistic Regression are the best performing classifiers. A small MLP (2 layers, with hidden width 256) also does well; tree-based approaches and kNN do worse. AEF does not benefit from regularization but does benefit from more complex classifiers like the MLP and random forest (Table 1).

3.1.2 Embedding quality as a function of ViT-patch size

Our goal is to produce 10m embeddings. The Attention Pooler allows us to do this with variable patch sizes for the encoder, controlling the computational cost of producing these embeddings. Figure 6 shows the cost-performance tradeoff of running OlmoEarth v1.3's encoder at patch size 1, 2 and 4 on the PASTIS val set. Running the encoder at patch size 2 dramatically reduces compute cost compared to patch size 1 (a **7.8** × decrease, from 2,396 GMACs to 308GMACs) with only a small decrease in performance (from 60.2 mIoU to 59.8 mIoU). This is the setting we use for evaluations (as discussed in Section 3.1) and for deployment.

3.2 Non-embedding evaluations

We also evaluate our model using the same evals as in OlmoEarth v1 and v1.2; this allows us to assess the overall strength of the backbone and Attention Pooler. When running these evaluations, we consider two methods of running OlmoEarth v1.3; with and without the Attention Pooler. When evaluating without the Attention Pooler, we average the Vision Transformer output tokens (following OlmoEarth v1.2).

The cost of the Attention Pooler is highly dependant on the evaluation being run (and – most importantly – on the number of time and modality tokens being compressed). For PASTIS (12 timesteps are compressed into a single token), the Attention Pooler only adds 2% to the compute cost. For m-eurosat or m-bigearthnet (the worst case – single timestep, single modality tasks where no compression happens at all), the Attention Pooler adds 16% to the compute cost. Averaged across all non-embedding eval tasks, the Attention Pooler adds 9% to the compute cost compared to an unmodified Vision Transformer backbone.

For these tasks, we use a patch size of 4 for both the encoder and Attention Pooler (unlike the embeddings, which are run at an encoder patch size of 2).

3.2.1 kNN and Linear Probing Results

		m-bigearthnet	m-so2sat	m-brick-kiln	m-forestnet	m-eurosat	BreizhCrops	CropHarvest-PRC	CropHarvest-PRC	CropHarvest-PRC	CropHarvest-Togo	CropHarvest-Togo	CropHarvest-Togo	m-cashewplant	m-SA-crop-type	PASTIS	PASTIS	MADOS	Sen1Floods11	AWF	AWF	AWF	Nandi	Nandi	Nandi
	Modalities	S2	S2	S2	L8	S2	S2	S1	S2	S1,S2	S1	S2	S1,S2	S2	S2	S1	S2	S2	S1	L8	S1	S2	L8	S1	S2
	Time series	✘	✘	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✘	✘	✔	✔	✔	✔	✔	✔
	Method	kNN	kNN	kNN	kNN	kNN	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	kNN	kNN	kNN	kNN	kNN	kNN
Model	Metric	μF1	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	mIOU	mIOU	mIOU	mIOU	mIOU	mIOU	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.
Best non-OlmoEarth		64.9	60.5	94.1	**52.3**	95.0	69.7	60.9	**81.9**	**79.3**	79.4	81.4	81.0	**54.1**	31.7	28.0	44.7	**68.4**	79.4	66.0	67.5	**79.0**	61.5	26.8	68.2
OlmoEarth v1	ViT Base	62.4	**67.7**	94.3	41.9	**95.5**	70.9	56.8	73.4	75.4	**80.1**	**87.3**	**82.0**	32.3	28.9	29.7	50.6	67.2	79.2	**77.0**	**68.5**	77.5	**67.9**	26.5	74.7
OlmoEarth v1.2	ViT Base	65.5	67.3	94.7	45.4	95.0	**73.3**	58.2	76.7	77.7	74.8	81.4	81.0	29.6	32.0	31.1	54.2	67.2	78.9	74.5	68.0	75.0	65.7	28.1	73.9
OlmoEarth v1.3	ViT Base + P	65.9	55.4	95.3	37.3	94.1	70.9	57.1	76.5	76.9	74.5	77.1	79.7	28.9	27.7	31.4	**57.5**	51.7	**79.5**	73.0	65.0	76.0	66.4	27.0	71.5
OlmoEarth v1.3	ViT Base	**66.1**	58.2	**96.8**	41.7	93.4	73.2	**62.4**	77.7	78.3	72.9	82.0	80.1	29.1	**32.8**	**32.6**	54.4	67.1	79.3	72.0	67.0	76.0	67.4	**28.5**	**77.6**

Table 3 kNN/Linear probe results on research benchmarks and real-world partner tasks for the OlmoEarth Base models. "Best non-OlmoEarth" is, for each task, the best score among all other models we evaluate (full results in Table 6). We run v1.3 with the Attention Pooler ("+P") and without. We **bold** the best score among the rows shown and highlight which ViT Base model wins between v1, v1.2, and v1.3.

When evaluated on the kNN/LP evaluations, OlmoEarth v1.3 performs slightly better than v1 and v1.2. When ranking across the 13 models which have results on all 24 tasks (AnySat, Clay, Panopticon and all OlmoEarth models), v1.3 Base – without the Attention Pooler – has the highest rank, 4.1 compared to 4.3 for OlmoEarth v1.2 and 4.2 for OlmoEarth v1 (Table 3). The same is true if we rank all models on the 13 Sentinel-2 tasks which all models support (Figure 7).

Adding the Attention Pooler hurts OlmoEarth v1.3 on these evaluations (by 2.1 points on average). The Attention Pooler does best when there are tokens to aggregate, and spatial context to leverage in the latent-self-attention layer - it does well on PASTIS-S2, which meets both of these requirements. We hypothesize this explains the Attention Pooler's regression overall, since many tasks are missing one of these two components. For example, we observe big drops on single-timestep single-modality tasks with nothing to aggregate (most significantly on MADOS) or tasks with only a single spatial token (CropHarvest, Nandi).

As discussed above (Section 3.2) the Attention Pooler adds a non-neglible cost to forward-passes - this is visible in the increased MACs for Base + P in Figure 7. However, we note again that many of these tasks represent worst-case scenarios for the Attention Pooler's cost, and the Attention Pooler's relative cost for our embeddings evaluations (Section 3.1) is much smaller.

3.2.2 Finetuning Results

OlmoEarth v1.3 is a strong finetuning model (Table 4). If we rank all models over all tasks (omitting missing task-model combinations), OlmoEarth v1.3 (with no Attention Pooling) has a mean rank of 4.34. This is behind OlmoEarth v1.2 Base (4.02) and ahead of OlmoEarth v1 Base (4.74). All three of these models are ahead of the non-OlmoEarth models (next best is Galileo-Base at 9.61) and of non-Base OlmoEarth models (next best is OlmoEarth v1.2 Small at 4.80). Like in the kNN / LP case, Attention Pooling penalizes the model, lowering its mean rank to 5.88. Attention Pooling tends to hurt most on single-image, single-modality tasks with no tokens to pool (m-so2sat, the partner vessel detection, length and type tasks).

v1.3 achieves performs well on both research tasks and the partner tasks, ranking second on each (mean ranks 4.50 and 4.23). In contrast, v1.2 is best on the research benchmarks (2.60) but fourth on the partner tasks (4.97) while the oppposite is true for v1 (first on partner tasks at 3.03, sixth on research benchmarks at 7.30).

3.3 Summary

OlmoEarth v1.3 produces state-of-the-art embeddings at 128 and 64 dimensions while remaining a strong encoder backbone for frozen and finetuned evaluations.

In spite of the additional parameters introduced by the Attention Pooler (as well as from the supervision heads and the distillation losses' MLPs), OlmoEarth v1.3 is considerably cheaper to train than OlmoEarth v1 or v1.2 while matching the previous models' performances.

The Attention Pooler lets us produce 10m embeddings at different encoder patch sizes, balancing performance and inference cost. Encoder patch size 2 keeps nearly all of patch size 1's performance while reducing inference

		m-bigearthnet	m-so2sat	m-brick-kiln	m-forestnet	m-eurosat	m-cashewplant	m-SA-crop-type	PASTIS	MADOS	Sen1Floods11	AWF	AWF	GEA North Africa	Forest Loss Driver	Live Fuel Moisture Content	Live Fuel Moisture Content	Mangrove	Mangrove	Nandi	Nandi	Vessel Detection	Vessel Detection	Vessel Length	Vessel Type	Solar Farm Detection
	Modalities	S2	S2	S2	L8	S2	S2	S2	S2	S2	S1	S2	S2, S1	S2	S2	S2	S2, S1	S2	S2, S1	S2	S2, S1	L8	S1	S2	S2	S2
	Time series	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘	✘	✔
Model	Metric	μF1	Acc.	Acc.	Acc.	Acc.	mIOU	mIOU	mIOU	mIOU	mIOU	Acc.	Acc.	Acc.	Acc.	L1	L1	Acc.	Acc.	Acc.	Acc.	F1	F1	L1	Acc.	mIoU
Anysat	ViT Base	68.4	56.7	98.7	51.6	95.9	80.4	34.2	60.9	63.3	77.4	78.0	83.0	59.3	84.6	19.1	19.4	96.9	97.1	78.4	76.7	-	-	73.4	43.5	82.6
Clay	ViT Large	65.7	61.3	98.2	49.2	95.1	73.9	33.4	48.9	68.9	78.5	77.0	-	53.4	93.3	24.8	-	96.6	-	30.9	-	70.7	79.9	16.4	68.3	82.2
CopernicusFM	ViT Base	71.3	66.8	98.3	-	98.4	78.7	33.6	54.6	66.0	78.6	79.0	79.0	58.8	90.0	25.2	24.5	97.1	97.1	68.2	55.9	-	77.4	16.7	69.6	77.6
CROMA	ViT Base	69.5	59.1	98.7	-	95.6	46.4	34.8	56.3	66.6	79.4	76.5	75.5	57.5	93.2	24.3	24.0	96.4	96.3	62.2	67.4	-	-	19.8	64.4	79.5
CROMA	ViT Large	71.8	58.9	97.8	-	97.5	47.8	36.0	58.1	68.8	79.4	79.5	-	56.6	92.3	24.6	24.1	96.6	96.2	76.4	-	-	-	-	-	81.7
DINOv3 Sat	ViT Large	69.9	63.3	98.9	**59.2**	96.7	80.6	34.5	42.8	64.7	-	34.5	-	43.0	80.4	90.2	-	65.8	-	35.8	-	-	-	30.3	54.8	70.7
Galileo	ViT Base	69.2	64.7	98.3	-	97.8	78.8	35.7	61.2	71.9	79.7	81.0	81.5	62.9	95.1	20.1	18.7	97.3	97.5	**81.9**	81.9	-	78.7	16.4	73.0	83.1
Panopticon	ViT Base	69.3	65.4	99.0	56.0	98.2	79.7	33.4	54.4	72.8	79.1	75.5	78.5	54.3	96.4	24.5	23.7	97.1	97.4	65.2	69.5	74.9	76.7	17.7	69.4	81.8
Prithvi v2	ViT Huge	70.6	64.7	98.2	-	96.8	81.1	38.8	58.6	69.3	-	80.0	-	60.6	92.4	-	-	97.2	-	77.1	-	71.1	-	17.4	68.2	84.1
Satlas	Swin Base	72.7	65.1	98.7	56.0	97.0	77.0	37.8	57.4	60.5	78.5	78.0	-	56.1	63.3	25.0	24.6	96.6	-	47.6	-	-	-	16.2	71.6	83.3
TerraMind	ViT Base	72.6	66.1	98.5	-	97.6	80.9	39.2	59.9	73.2	79.5	84.0	82.0	49.8	96.4	24.3	23.8	**97.7**	96.8	66.1	79.3	-	79.6	18.1	-	83.5
TerraMind	ViT Large	**74.0**	65.4	98.1	-	97.8	81.3	41.1	60.9	71.5	79.5	81.5	-	51.1	93.9	24.5	24.5	96.5	96.9	66.1	-	-	-	-	-	83.0
OlmoEarth v1	ViT Nano	66.8	61.5	98.2	50.3	95.7	39.5	35.4	53.0	60.6	78.8	82.5	82.5	61.1	96.0	20.4	19.7	97.4	97.4	75.6	74.8	70.2	75.5	17.1	72.0	82.1
OlmoEarth v1	ViT Tiny	69.6	63.5	98.7	53.2	97.9	72.5	38.5	60.3	71.5	79.7	85.0	85.5	60.6	97.7	19.8	19.2	97.6	97.7	78.2	76.4	74.4	76.9	15.8	73.5	85.2
OlmoEarth v1	ViT Base	72.0	68.6	98.2	51.2	**98.6**	79.8	39.6	64.3	77.8	79.8	**87.0**	**86.0**	62.4	97.1	18.5	**17.9**	97.6	**97.9**	81.8	82.2	75.4	79.2	15.4	74.6	85.4
OlmoEarth v1	ViT Large	72.4	68.1	98.8	52.7	98.3	80.6	40.8	**66.3**	**81.8**	79.8	84.5	-	58.8	**97.9**	19.9	18.5	97.6	97.6	81.0	-	-	-	-	-	84.2
OlmoEarth v1.2	ViT Nano	68.0	60.8	98.1	53.7	96.5	54.0	36.3	52.7	58.1	78.9	75	75.5	61.5	95.4	20.4	19.9	96.3	96.6	79.2	78.1	71.1	75.1	16.8	72.4	80.1
OlmoEarth v1.2	ViT Tiny	71.3	67.4	98.8	56.8	98.1	87.8	41.7	63.7	67.7	79.4	83.5	84.5	59.3	97.8	19.5	18.8	97.2	97.5	80.7	81.1	75.2	76.9	14.3	73.8	86.6
OlmoEarth v1.2	ViT Small	72.0	66.5	99.0	56.5	98.2	**95.8**	44.0	64.7	69.7	79.8	83.0	82.0	62.0	**97.9**	19.4	19.0	97.5	97.7	78.3	80.2	**75.6**	**81.9**	**14.2**	**75.9**	**87.5**
OlmoEarth v1.2	ViT Base	73.0	**70.8**	**99.1**	56.1	98.3	82.5	43.5	65.8	75.8	**80.5**	83.0	85.0	60.2	97.1	19.1	18.3	97.5	97.6	81.6	**82.5**	74.8	78.3	14.5	**75.9**	87.3
OlmoEarth v1.3	ViT Base + P	73.3	66.5	99.0	55.3	98.0	80.5	**44.4**	65.7	72.3	79.7	84.5	79.5	62.9	97.7	**18.4**	18.5	97.5	97.7	78.7	80.7	71.9	79.0	16.2	70.6	84.8
OlmoEarth v1.3	ViT Base	72.7	68.4	99.0	52.7	98.5	81.0	42.7	64.1	74.6	**80.5**	86.5	85.5	**65.6**	97.3	19.1	18.1	97.5	97.8	77.3	78.3	74.7	79.5	14.6	75.7	**87.5**

Table 4 Finetuning results on research benchmarks and partner tasks. Due to resource constraints, we do not fine-tune large models on all tasks. We run v1.3 with the Attention Pooler ("+P") and without. We highlight which ViT Base model wins between v1, v1.2, and v1.3.

costs by 7.8×. This is the configuration we use to compute global embeddings, described below.

4 Creating & Serving Global Scale Embeddings

Computing embeddings over the whole Earth is expensive. Alongside the OlmoEarth v1.3 weights we release a precomputed embedding dataset, readable directly from object storage: 128-dimensional int8 vectors at 10 m resolution, one per pixel per reference year.

Like the evaluations in Section 3.1, embeddings are computed using Sentinel-1, Sentinel-2 and Landsat-8/9. Each year is split into twelve 30-day periods ending on January 1 of the following year, and for each modality and period the scenes covering a window are composited into one mosaic by first valid pixel, least cloudy first for Sentinel-2 and Landsat, giving 12 timesteps per modality.

Embeddings are annual, each computed from one calendar year of imagery. A Zarr array's shape is fixed when it is created, so the store holds one slot per year from 2017 to 2025 from the start; each year is filled independently, and a year not yet computed takes no storage. The coverage area is the union of three things: land; full-resolution shorelines, which keep barrier islands and atolls; and water that all three modalities observe consistently, which we take to be at least 11 months in at least 7 of the years 2017–2025 and which selects most of the coastal zone. A window touching that area is then computed in full, water included. The grid follows the UTM system, which spans 80°S to 84°N. Within it, the coverage area runs from 59.5°S to 83.6°N and does not include Antarctica.

Embeddings for 2025, 2024 and 2023 are global, and for 2017–2022 cover select countries and are in progress over the same extent. We are still computing embeddings, so the coverage described here is a snapshot; the dataset publishes a coverage layer of the extent each year will reach when complete.

4.1 Inference at scale

Inference is embarrassingly parallel over space and time, so the pipeline is organized as a work queue. We tile each UTM zone, discard tiles that fall outside the zone or touch none of the coverage area, and enqueue the rest onto a Beaker [10] queue as independent work units, each an 8192×8192 pixel block for one reference year. A block is split into 2048×2048 windows, and every window that touches the coverage area is computed

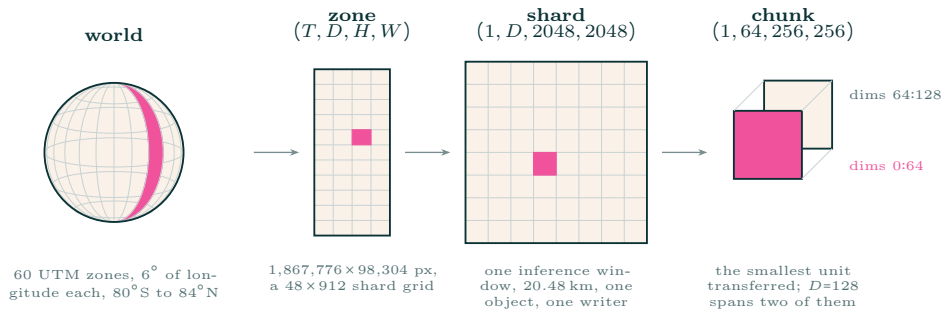


Figure 8 Layout of the OlmoEarth embedding dataset. Each UTM zone is one Zarr array over (reference year, embedding dimension, northing, easting) at 10 m, sparse so unwritten regions cost nothing. A *shard* is the unit of storage and is pinned to one inference window, so a window writes exactly one object and concurrent writers never contend. A *chunk* is the unit of transfer; chunking the embedding axis at 64 means a Matryoshka read of the first 64 dimensions fetches one chunk rather than two.

in full, water included. A supervisor keeps the queue filled and the worker pool at its target size; workers claim units, materialize the imagery they need, run the model, and write results directly into the store.

The model runs at patch size 2 with 10 m output (Section 2). Each window is split into 480 × 480 pixel crops, each a single forward pass; adjacent crops overlap by 32 pixels and each keeps 16 of them at a shared edge. Workers run on NVIDIA H100 nodes and complete a full unit in a median of 30 minutes, or roughly 0.5 GPU-hours per unit. To add capacity we also run workers on NVIDIA A100 (40 GB) virtual machines on Google Cloud, which take units from the same queue and write to the same store. A 480-pixel crop does not fit in 40 GB, so these workers use 384-pixel crops; on the same windows their output matches the H100 output at a mean cosine similarity of 0.9999. An A100 completes a full unit in a median of 84 minutes. Throughput scales with the size of the pool.

Each work unit is idempotent: a completion marker is written only after its embeddings land, and enumeration skips units already marked, so the build is interruptible and resumable. Over a multi-day run workers are preempted and occasionally hang, and none of that costs more than the units a worker holds at the time.

Materializing a unit's imagery is network-bound and takes minutes, so each worker overlaps it with inference. The queue lets a worker hold two units at once: while the GPU runs one, a separate process downloads the next unit's imagery to local disk, and inference on that unit then reads from disk. Only download time that outlasts the inference ahead of it adds to a unit's time. On H100 workers, a unit's time grows with the number of windows it writes, at 1.9 minutes per window plus about 2 minutes per unit.

4.2 Removing tiling artifacts using Searchlight



S2 Tiled Inference Searchlight Inference

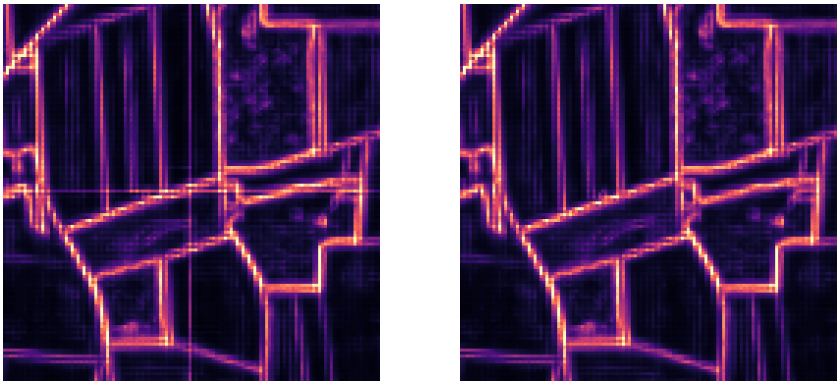
Figure 9 Tiled inference – even with significant overlap between crops – introduces artifacts between crops, especially on homogeneous areas like water. These artifacts disappear when using searchlight inference.

A common challenge when producing maps from ViT-based models is tiling artifacts. Pixels at the edge of input crops have less spatial context, and that context is biased in a certain direction; this can create discrepancies between neighbouring pixels which were processed by different crops (Figure 9).

To address this issue, we run inference using Searchlight, producing artifact-free embeddings. Searchlight inference uses neighborhood attention [11], which lets it run on inputs much larger than those seen during training, since compute cost grows only linearly with input size. It also reduces tiling artifacts by making inference translation-invariant: apart from boundary effects, the model applies exactly the same operation at every pixel. Instead of passing batches of crops to the model, we instead pass a single, large "domain" to the model. We then apply an attention mask such that every pixel sees the 16×16 window centered around it, effectively mirroring the case where it is the center pixel of a 16×16 window.

Searchlight introduces attention (neighborhood vs full) and input size (large vs small) discrepancies between training and inference. To ensure this does not negatively affect embedding quality, we measure PASTIS performance using tiled inference and Searchlight inference. PASTIS validation mIoU remains stable (we obtained 60.4 using Searchlight and 59.8 using tiled inference).

4.3 Seamless windows



No margin 48 px margin

Figure 10 Dissimilarity between adjacent pixels where four inference windows meet (bright is a boundary). Without a margin the window edges show as a faint cross; with a 48-pixel margin they disappear and field boundaries are unchanged.

Searchlight removes seams between crops, but each 2048×2048 window is still run on its own, so pixels at its edge see context from one side only and a faint seam remains every 20.48 km (Figure 10). We therefore materialize and run each window with a 48-pixel margin on every side and write only its central 2048×2048 pixels, so shards and the write path are unchanged. The mean cosine similarity between pixels adjacent across a window edge, relative to adjacent pixels just inside it, rises from 0.960 to 0.994 (median over six edges at three sites), while window interiors are unchanged (cosine similarity ≥ 0.9999). A 112-pixel margin gives no further gain, and 48 pixels adds 9.6% to the pixels computed per window. 2025, most of 2024 and a small part of 2019 and 2023 were built before the change, so each block records its margin in its completion marker and provenance.

4.4 Dataset layout

The embeddings are a Zarr v3 store with one array per UTM zone over $(\text{year}, \text{dimension}, \text{northing}, \text{easting})$. It uses the sharding codec, which packs many chunks into a single storage object. Each zone is held in its northern CRS (EPSG:326NN) with a northing axis continuing below zero south of the equator, so one array spans the full UTM range, 80°S to 84°N, with coordinates monotonic across the equator (Figure 8). The conventional alternative pairs each northern CRS with a southern one (EPSG:327NN) whose northing restarts at 10,000 km at the equator, so any read crossing it would have to join two arrays.

Shard size is pinned to the inference window at 2048 × 2048 pixels, which makes the write path lock-free by construction: one window produces exactly one object, owned by exactly one worker, so hundreds of writers proceed with no coordination and no partial-object states.

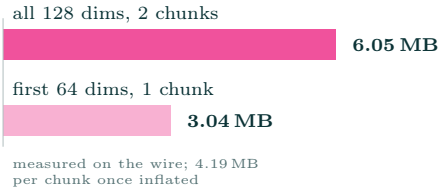


Figure 11 Bytes transferred to read one 256×256 tile of embeddings. Because a whole chunk is the smallest unit the store can move, reading the Matryoshka $[\ldots, :64]$ prefix costs half of the full 128 dimensions rather than the same.

Within a shard we chunk at 256 × 256 pixels under Zstandard level 3, chosen by measuring bytes moved over the network across a sweep of layout variants. A chunk is the smallest unit the store can transfer, and compressed chunks are not seekable, so a single-pixel query still moves the whole chunk containing it, roughly 3 MB. That is cheap occasionally but wasteful in bulk, so the store is built for area reads.

Chunking also decides whether a nested embedding [16] is cheaper to read and not just to store: a prefix saves bytes only when the dimensions beyond it sit in a different chunk. We chunk the embedding axis at 64, the narrowest width the model is trained to emit (Section 2.3), so reading $[\ldots, :64]$ fetches one chunk instead of two and moves 3.04 MB instead of 6.05 MB per tile (Figure 11).

The store follows the `geoemb` embeddings-zarr convention,[4] and records model identity, sources and quantization parameters as attributes, so any byte is traceable to the code that produced it.

4.5 Quantization

Embeddings are stored as int8 using a signed-power scheme [3]. Writing $s = 127.5$ for the scale and $\gamma = \frac{1}{2}$ for the exponent, a float value x is encoded and recovered by

$$q = \left\lceil \left\lfloor s \, \mathrm{sgn}(x) \, |x|^{\gamma} \right\rceil \right\rceil_{-127}^{127}, \qquad \hat{x} = \mathrm{sgn}(q) \left| \frac{q}{s} \right|^{1/\gamma}, \tag{1}$$

where $\lfloor \cdot \rceil$ rounds to the nearest integer and $\lceil \cdot \rceil_a^b$ clamps to $[a, b]$; the value -128 is reserved as nodata. We quantize the output as emitted rather than L2-normalizing it, which would discard magnitude that downstream classifiers use; in practice nearly all coordinates of the student head's LayerNorm output fall inside the scheme's $[-1, 1]$ range, and the clamp handles the rest. Median cosine similarity between a float embedding and its dequantized form is 0.99997.

4.6 Provenance

Each block writes a provenance record beside its completion marker. For every window and modality it lists the scenes composited into each 30-day mosaic, in compositing order. Each record also carries the window's materialized bounds and its margin. Each scene is resolved to the public source its imagery was read from: Sentinel-1 RTC and Sentinel-2 L2A from Microsoft Planetary Computer, and Landsat Collection 2 from Google Cloud's public Landsat bucket, with its ESA or USGS product name. We release these records as GeoParquet tables per reference year: one row per window, one per scene in each 30-day mosaic, and one per scene with its acquisition time and footprint. For 2025 that is about 450,000 windows, 21 million mosaic entries and 730,000 scenes, in 150 MB.

4.7 Distribution

Hosting and serving the dataset is a significant challenge in its own right. At 10 m, 128 int8 dimensions per pixel come to roughly 150 TB per reference year at global coverage, well over a petabyte across 2017 to 2025, and the dataset is meant to be read directly from object storage by anyone, at any scale. We are very grateful to Source Cooperative, which hosts the dataset and serves it openly to the community.[5]

[4] https://github.com/geo-embeddings/embeddings-zarr-convention

[5] https://source.coop/olmoearth/olmoearth-v1-3-embeddings

The store is read directly through Source Cooperative's S3-compatible endpoint with any Zarr client, with no API key or tile server needed. Worked examples ship with the dataset.

4.8 Interactive exploration

We also publish a browser application at `https://olmoearth.allenai.org/embeddings`, which shows the false-color layer of Section 4.9 as a web map and runs simple analyses on the embeddings themselves, such as finding places similar to one the user clicks or clustering an area they draw. Any result can be exported as a Python script that reproduces it against the dataset.

4.9 False-color visualization

Embeddings are not directly viewable, so the explorer shows a false-color rendering: a PCA basis fitted on a stratified sample, projected to three components as a uint8 RGB raster. Three components capture only a fifth to two fifths of local variance, so this is a visualization and not a reduced-dimension version of the embeddings. It is not part of the released dataset. It lives in its own store, reprojected from the embeddings' UTM grid to Web Mercator (EPSG:3857) on the standard XYZ scheme, so a pyramid level is a web-map zoom level and a client reads a bounded number of chunks at any extent.

5 Environmental Impact

5.1 Pretraining

We use the same technique as OlmoEarth v1 and OlmoEarth v1.2 to measure the pretraining cost of the OlmoEarth v1.3 model (Table 5). As with OlmoEarth v1 and OlmoEarth v1.2, this estimate represents a lower bound since it does not account for hardware manufacturing, transportation, etc. We significantly decreased the pretraining cost of OlmoEarth v1.3 compared to OlmoEarth v1.2 (and therefore OlmoEarth v1). Compared to OlmoEarth v1 / v1.2, pretraining OlmoEarth v1.3 Base required 18% / 53% of the GPU hours, and emitted 22% / 50% of the tCO_2eq.

Model	Size	Hardware	GPU Hrs	Energy (kWh)	Carbon (tCO_2eq)	Water (kL)
OlmoEarth v1	Nano	H100	1,149	195	0.08	0.30
OlmoEarth v1	Tiny	H100	1,149	205	0.08	0.32
OlmoEarth v1	Base	H100	2,989	803	0.32	1.24
OlmoEarth v1.2	Nano	H100	1,151	191	0.09	0.30
OlmoEarth v1.2	Tiny	H100	1,132	225	0.10	0.35
OlmoEarth v1.2	Small	H100	1,253	265	0.12	0.41
OlmoEarth v1.2	Base	H100	1,012	317	0.14	0.49
OlmoEarth v1.3	Base	H100	539	161	0.07	0.25

Table 5 Approximate environmental impact of pretraining OlmoEarth v1, v1.2 and v1.3.

5.2 Computing embeddings

Computing global annualized embeddings dwarfs OlmoEarth v1.3's pretraining costs. One year of global embeddings costs approximately 14,000 H100 hours (as noted in Section 4.1, our inference mixes H100s and A100s, so this is an estimate). This is ≈ 26× the pretraining cost per year of annualized embeddings.

We selected patch size 2 for inference as a balance between performance and compute cost (Section 3.1.2). Anyone generating embeddings locally who is limited by compute can produce 10m embeddings using an encoder patch size of 4 for roughly 5× less compute, at the cost of slightly lower performance (Pastis validation mIoU drops from 59.8 at patch size 2 to 58.6 at patch size 4).

6 Discussion

Our mission with OlmoEarth is to make it easy to get high quality insights from Earth Observations. Embeddings play an important role in this mission, allowing practitioners to quickly and cheaply explore our changing planet. OlmoEarth v1.3 contributes in two ways; by halving pretraining costs, we hope more researchers can investigate geospatial foundation modelling, and by precomputing embeddings we hope it becomes easier for anyone to make high quality maps without running the model at all.

We release annualized embeddings at a 10 m resolution: global for 2023, 2024 &–2025, and for select countries for 2017–2022. Users can download either the full 128-dimensional embeddings, or the 64-dimensional prefix. Many tasks require more specific time ranges; for such tasks, we point users to OlmoEarth Studio (`https://olmoearth.allenai.org/platform`), which runs the same model over user-chosen time windows. OlmoEarth v1.3 can also be run locally to make embeddings independently on local compute.

We release OlmoEarth v1.3 and the embeddings under the same "OlmoEarth Artifact License" as OlmoEarth v1 and v1.2. This open license restricts use for military, defense-related, and extractive industry applications.

References

[1] Guillaume Astruc, Nicolas Gonthier, Clement Mallet, and Loic Landrieu. AnySat: An Earth observation model for any resolutions, scales, and modalities. *arXiv preprint arXiv:2412.14123*, 2024.

[2] Favyen Bastani, Piper Wolters, Ritwik Gupta, Joe Ferdinando, and Aniruddha Kembhavi. SatlasPretrain: A large-scale dataset for remote sensing image understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16772–16782, 2023.

[3] Christopher F. Brown, Michal R. Kazmierski, Valerie J. Pasquarella, William J. Rucklidge, Masha Samsikova, Chenhui Zhang, Evan Shelhamer, Estefania Lahera, Olivia Wiles, Simon Ilyushchenko, Noel Gorelick, Lihui Lydia Zhang, Sophia Alj, Emily Schechter, Sean Askay, Oliver Guinan, Rebecca Moore, Alexis Boukouvalas, and Pushmeet Kohli. AlphaEarth Foundations: An embedding field model for accurate and efficient global mapping from sparse label data, 2025.

[4] Clay. Clay Foundation Model - Clay Foundation Model. https://clay-foundation.github.io/model/, 2025.

[5] Etalab. Registre parcellaire graphique (RPG) : contours des parcelles et îlots culturaux et leur groupe de cultures majoritaire. https://www.data.gouv.fr/en/datasets/registre-parcellaire-graphique-rpg-contours-des-parcelles-et-ilots-culturaux-et-leur-groupe-de-cultures-majori 2017. Accessed 2026-09-03.

[6] Zhengpeng Feng, Sadiq Jaffer, Ira Shokar, Jovana Knezevic, James Ball, Pedro Sousa, Mark Elvers, Madeline Lisaius, Clement Atzberger, Robin Young, et al. Tessera v2: Scaling pixel-wise earth foundation models. *arXiv preprint arXiv:2607.03949*, 2026.

[7] Anthony Fuller, Koreen Millard, and James Green. CROMA: Remote sensing representations with contrastive radar-optical masked autoencoders. In *Advances in Neural Information Processing Systems*, 2024.

[8] Vivien Sainte Fare Garnot and Loic Landrieu. Panoptic segmentation of satellite image time series with convolutional temporal attention networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4872–4881, 2021.

[9] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote sensing of Environment*, 202:18–27, 2017.

[10] M. Guerquin. Introducing ai2's beaker. https://web.archive.org/web/20241231204439/https://medium.com/ai2-blog/beaker-ed617d5f4593, 2022. Accessed: 2025-05-15.

[11] Ali Hassani, Fengzhe Zhou, Aditya Kane, Jiannan Huang, Chieh-Yun Chen, Min Shi, Steven Walton, Markus Hoehnerbach, Vijay Thakkar, Mikhail Isaev, et al. Generalized neighborhood attention: Multi-dimensional sparse attention at the speed of light. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

[12] Henry Herzog, Favyen Bastani, Yawen Zhang, Gabriel Tseng, Joseph Redmon, Hadrien Sablon, Ryan Park, Jacob Morrison, Alexandra Buraczynski, Karen Farley, et al. Olmoearth: Stable latent image modeling for multimodal earth observation. In *CVPR*, 2026.

[13] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*. PMLR, 2021.

[14] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022.

[15] Johannes Jakubik, Felix Yang, Benedikt Blumenstiel, Erik Scheurer, Rocco Sedona, Stefano Maurogiovanni, Jente Bosmans, Nikolaos Dionelis, Valerio Marsocci, Niklas Kopp, Rahul Ramachandran, Paolo Fraccaro, Thomas Brunschwiler, Gabriele Cavallaro, Juan Bernabe-Moreno, and Nicolas Longépé. TerraMind: Large-scale generative multimodality for earth observation. In *ICCV*, 2025.

[16] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In *Advances in Neural Information Processing Systems*, 2022.

[17] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR, 2019.

[18] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*. PmLR, 2023.

[19] Yuchi Ma, Yawen Shen, Anu Swatantran, and David B Lobell. Harvesting alphaearth: Benchmarking the geospatial foundation model for agricultural downstream tasks. *International Journal of Applied Earth Observation and Geoinformation*, 2026.

[20] National Aeronautics and Space Administration (NASA) Earthdata. Shuttle Radar Topography Mission. https://e4ftl01.cr.usgs.gov/MEASURES/SRTMGL1.003/, 2018.

[21] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[22] Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3, 2025.

[23] Daniela Szwarcman, Sujit Roy, Paolo Fraccaro, Thorsteinn Elí Gíslason, Benedikt Blumenstiel, Rinki Ghosal, Pedro Henrique de Oliveira, Joao Lucas de Sousa Almeida, Rocco Sedona, Yanghui Kang, et al. Prithvi-eo-2.0: A versatile multi-temporal foundation model for earth observation applications. *arXiv preprint arXiv:2412.02732*, 2024.

[24] Jamie Tolan, Hung-I Yang, Benjamin Nosarzewski, Guillaume Couairon, Huy V Vo, John Brandt, Justine Spore, Sayantan Majumdar, Daniel Haziza, Janaki Vamaraju, et al. Very high resolution canopy height maps from rgb imagery using self-supervised vision transformer and convolutional decoder trained on aerial lidar. *Remote Sensing of Environment*, 300:113888, 2024.

[25] Gabriel Tseng, Ruben Cartuyvels, Ivan Zvonkov, Mirali Purohit, David Rolnick, and Hannah Kerner. Lightweight, pre-trained transformers for remote sensing timeseries. *arXiv preprint arXiv:2304.14065*, 2023.

[26] Gabriel Tseng, Anthony Fuller, Marlena Reil, Henry Herzog, Patrick Beukema, Favyen Bastani, James R Green, Evan Shelhamer, Hannah Kerner, and David Rolnick. Galileo: Learning global & local features of many remote sensing modalities. In *Forty-second International Conference on Machine Learning*, 2025.

[27] Gabriel Tseng, Yawen Zhang, Favyen Bastani, Henry Herzog, Joseph Redmon, Hadrien Sablon, Piper Wolters, Ando Shah, Patrick Alan Johnson, Christopher Wilhelm, et al. Olmoearth v1. 2: A more efficient family of olmoearth models. *arXiv preprint arXiv:2605.20804*, 2026.

[28] United States Department of Agriculture (USDA) National Agricultural Statistics Service (NASS). Cropland Data Layer: USDA NASS, 2024. National Agricultural Statistics Service Marketing and Information Services Office, Washington, D.C. Retrieved from Link: https://croplandcros.scinet.usda.gov/.

[29] Kristof Van Tricht, Jeroen Degerickx, Sven Gilliams, Daniele Zanaga, Marjorie Battude, Alex Grosu, Joost Brombacher, Myroslava Lesiv, Juan Carlos Laso Bayas, Santosh Karanam, et al. WorldCereal: a dynamic open-source system for global-scale, seasonal, and reproducible crop and irrigation mapping. *Earth System Science Data Discussions*, 2023:1–36, 2023.

[30] Leonard Waldmann, Ando Shah, Yi Wang, Nils Lehmann, Adam Stewart, Zhitong Xiong, Xiao Xiang Zhu, Stefan Bauer, and John Chuang. Panopticon: Advancing any-sensor foundation models for earth observation. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR) Workshops*, 2025.

[31] Yi Wang, Zhitong Xiong, Chenying Liu, Adam J Stewart, Thomas Dujardin, Nikolaos Ioannis Bountos, Angelos Zavras, Franziska Gerken, Ioannis Papoutsis, Laura Leal-Taixé, et al. Towards a unified copernicus foundation model for earth vision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

[32] Piper Wolters, Gabriel Tseng, Yawen Zhang, Joseph Redmon, Favyen Bastani, Hadrien Sablon, and Ando Shah. PLANTEUR: A crop-type segmentation benchmark for france's overseas territories. Technical report, Allen Institute for AI, 2026. Dataset: https://huggingface.co/datasets/allenai/planteur-benchmark.

[33] Daniele Zanaga, Ruben Van De Kerchove, Dirk Daems, Wanda De Keersmaecker, Carsten Brockmann, Grit Kirches, Jan Wevers, Oliver Cartus, Maurizio Santoro, Steffen Fritz, et al. ESA WorldCover 10 m 2021 v200. *ESA WorldCover Project*, 2022.

A Full kNN / LP results

Table 3 compares OlmoEarth-Base models to "best non-OlmoEarth models". We share the full results in Table 6.

		m-bigearthnet	m-so2sat	m-brick-kiln	m-forestnet	m-eurosat	BreizhCrops	CropHarvest-PRC	CropHarvest-PRC	CropHarvest-PRC	CropHarvest-PRC	CropHarvest-Togo	CropHarvest-Togo	CropHarvest-Togo	m-cashewplant	m-SA-crop-type	PASTIS	PASTIS	MADOS	Sen1Floods11	AWF	AWF	AWF	Nandi	Nandi	Nandi
	Modalities	S2	S2	S2	L8	S2	S2	S1	S2	S1,S2	S1	S2	S1,S2	S2	S2	S2	S1	S2	S2	S1	L8	S1	S2	L8	S1	S2
	Time series	✘	✘	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✘	✘	✔	✔	✔	✔	✔	✔
	Method	kNN	kNN	kNN	kNN	kNN	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	LP	kNN	kNN	kNN	kNN	kNN	kNN
Model	Metric	μF1	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	mIOU	mIOU	mIOU	mIOU	mIOU	mIOU	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.
Anysat [1]	ViT Base	54.5	36.5	77.0	34.0	77.6	62.7	57.0	74.1	72.3	69.6	78.4	74.5	24.6	27.2	24.2	41.9	41.3	77.8	60.0	60.0	64.0	47.4	20.5	55.8
Clay [4]	ViT Large	48.8	38.7	92.8	40.3	86.1	57.0	56.7	66.5	63.4	78.4	67.0	67.3	30.8	23.1	19.9	22.6	47.5	78.9	59.5	61.5	56.5	52.2	23.4	57.0
CopernicusFM [31]	ViT Base	64.6	50.3	91.2	-	91.4	65.5	55.1	72.8	74.4	77.5	75.8	70.6	32.2	28.4	15.9	32.1	63.9	77.6	-	59.0	67.0	-	24.4	59.8
CROMA [7]	ViT Base	61.3	51.3	94.0	-	93.1	69.0	56.8	74.3	75.1	76.8	80.7	76.5	24.9	30.3	26.3	44.7	60.4	78.8	-	67.5	**79.0**	-	24.6	68.2
CROMA [7]	ViT Large	59.2	48.2	91.9	-	93.6	68.2	56.2	72.7	71.0	79.4	76.5	81.0	27.0	30.4	25.9	42.7	66.4	78.8	-	62.5	71.0	-	26.1	68.2
DINOv3 [22]	ViT Base	51.0	47.1	91.3	43.3	86.6	31.3	-	64.5	-	-	67.0	-	23.5	26.7	-	18.1	53.5	-	48.5	-	61.0	42.5	-	54.1
DINOv3 [22]	ViT Large	55.8	45.3	89.9	46.0	84.0	31.3	-	66.1	-	-	68.3	-	24.5	26.1	-	17.4	52.4	-	43.5	-	60.5	39.9	-	49.8
DINOv3 [22]	ViT Huge+	57.0	45.7	88.2	46.9	86.1	31.3	-	68.7	-	-	68.3	-	25.1	26.7	-	17.4	48.1	-	47.5	-	54.0	43.5	-	55.4
DINOv3 [22]	ViT 7B	60.8	46.6	91.3	48.0	85.2	31.3	-	68.7	-	-	70.9	-	34.3	27.9	-	21.1	52.2	-	47.5	-	57.0	44.7	-	56.7
DINOv3 Sat [22]	ViT Large	60.2	44.0	91.4	44.2	89.2	31.3	-	70.1	-	-	68.6	-	32.4	28.5	-	22.5	57.5	-	42.5	-	69.5	35.5	-	48.4
DINOv3 Sat [22]	ViT 7B	61.6	50.1	91.4	47.0	91.3	31.3	-	72.2	-	-	71.9	-	**54.1**	31.7	-	26.3	59.7	-	49.5	-	68.5	31.8	-	42.5
Galileo [26]	ViT Nano	55.0	53.7	91.6	-	91.5	66.3	60.9	74.4	72.4	75.2	70.3	78.1	21.2	19.5	19.2	19.1	53.1	78.6	-	65.5	67.5	-	24.4	64.2
Galileo [26]	ViT Tiny	55.8	53.1	90.7	-	93.1	66.7	55.7	80.3	79.3	69.9	78.8	77.1	23.6	21.5	23.4	27.7	61.1	78.6	-	65.5	71.0	-	25.0	66.7
Galileo [26]	ViT Base	58.3	55.7	88.5	-	94.3	69.7	60.9	**81.9**	79.3	67.3	80.1	77.5	28.9	25.3	28.0	39.6	**68.4**	79.4	-	66.5	72.5	-	26.8	67.3
Panopticon [30]	ViT Base	64.9	60.5	94.1	**52.3**	95.0	57.7	55.9	75.9	75.6	72.2	72.9	76.5	32.7	27.3	23.7	30.2	66.1	78.0	66.0	65.0	71.5	60.4	22.9	65.2
Presto [25]	ViT Nano	-	-	-	-	-	60.9	59.3	74.3	76.6	78.4	81.4	72.5	-	-	16.3	28.2	-	-	-	60.5	53.5	-	25.2	57.6
Prithvi v2 [23]	ViT Large	53.8	42.6	90.3	37.9	82.2	66.2	-	72.0	-	-	71.9	-	51.1	27.3	-	37.2	55.6	-	57.0	-	55.5	57.7	-	59.3
Prithvi v2 [23]	ViT Huge	52.1	39.5	89.9	41.4	81.2	66.1	-	71.7	-	-	69.9	-	49.2	29.7	-	37.5	58.2	-	59.0	-	55.0	57.9	-	55.9
Satlas [2]	Swin Base	52.3	44.5	91.9	36.9	89.1	64.6	57.4	71.3	-	76.8	75.8	-	30.6	24.0	10.5	14.4	30.2	72.9	57.5	52.0	62.0	61.5	25.1	60.2
TerraMind [15]	ViT Base	63.9	46.7	92.9	-	91.4	66.4	57.0	75.1	75.6	74.2	74.8	77.5	46.0	30.4	22.7	40.9	66.0	78.7	-	66.0	69.5	-	24.7	64.2
TerraMind [15]	ViT Large	63.9	47.4	92.9	-	90.9	68.2	56.2	74.7	72.0	75.2	77.8	75.2	50.4	31.2	22.3	41.3	67.5	78.4	-	62.0	67.0	-	23.3	65.3
OlmoEarth v1	ViT Nano	59.5	54.3	94.7	38.8	91.7	64.1	58.0	79.5	74.3	73.5	81.7	**83.7**	25.5	23.6	18.1	35.0	55.2	78.2	73.0	61.5	69.5	57.7	24.8	67.4
OlmoEarth v1	ViT Tiny	59.4	61.8	93.3	40.5	93.7	64.0	58.3	78.6	**80.3**	75.8	85.6	82.4	24.7	23.2	21.4	40.1	58.6	78.5	75.5	64.0	76.0	60.7	24.7	69.0
OlmoEarth v1	ViT Base	62.4	67.7	94.3	41.9	95.5	70.9	56.8	73.4	75.4	**80.1**	**87.3**	82.0	32.3	28.9	29.7	50.6	67.2	79.2	**77.0**	68.5	77.5	**67.9**	26.5	74.7
OlmoEarth v1	VIT Large	62.0	**68.2**	96.5	41.6	**96.4**	70.7	56.6	74.1	76.1	67.6	78.1	79.7	30.9	28.5	30.6	51.8	66.4	**79.8**	76.0	66.5	73.0	66.4	26.2	73.6
OlmoEarth v1.2	ViT Nano	59.5	55.3	94.8	38.5	91.4	67.5	**63.7**	77.5	80.2	73.9	79.4	82.7	26.8	22.4	22.5	35.9	58.1	78.7	71.5	64.0	71.5	66.9	25.9	71.0
OlmoEarth v1.2	ViT Tiny	61.7	58.8	92.6	39.9	93.8	70.6	59.2	77.6	80.0	74.5	82.0	79.4	27.1	24.4	24.9	45.3	63.0	78.7	74.0	64.5	76.0	**67.9**	28.4	67.3
OlmoEarth v1.2	ViT Small	64.8	65.7	91.8	44.4	94.5	**79.1**	59.0	79.1	78.1	73.5	83.0	80.4	28.5	26.7	28.3	50.7	61.3	79.3	**77.0**	**69.0**	74.5	67.2	**29.7**	73.2
OlmoEarth v1.2	ViT Base	65.5	67.3	94.7	45.4	95.0	73.3	58.2	76.7	77.7	74.8	81.4	81.0	29.6	32.0	31.1	54.2	67.2	78.9	74.5	68.0	75.0	65.7	28.1	73.9
OlmoEarth v1.3	ViT Base + P	65.9	55.4	95.3	37.3	94.1	70.9	57.1	76.5	76.9	74.5	77.1	79.7	28.9	27.7	31.4	**57.5**	51.7	79.5	73.0	65.0	76.0	66.4	27.0	71.5
OlmoEarth v1.3	ViT Base	**66.1**	58.2	**96.8**	41.7	93.4	73.2	62.4	77.7	78.3	72.9	82.0	80.1	29.1	**32.8**	**32.6**	54.4	67.1	79.3	72.0	67.0	76.0	67.4	28.5	**77.6**

Table 6 kNN/Linear probe results on research benchmarks and real-world partner tasks. Not all models can run on all tasks due to incompatible input modalities. We run v1.3 with Attention Pooling ("+P") and without. We highlight which ViT Base model wins between v1, v1.2, and v1.3.