(Full) Attention is More Than You Need: Sparse Cross-View Attention in VGGT

Maria Pilligua
EPFL CVLab
maria.pilligua@epfl.ch

Supervised by Aoxiang Fan and Pascal Fua

Abstract

Feed-forward models such as VGGT reconstruct cameras and geometry from many images in a single pass, but their global cross-view attention compares every patch to every patch across all frames, so its cost grows quadratically and caps how many views the model can take at once. Most of that attention is wasted: a patch only needs the few patches in other frames that see the same 3D point. Restricting attention to those correspondences makes it near-linear in the number of frames and, given accurate matches, even improves reconstruction over dense attention. The difficulty is circular, since finding correspondences across views is the all-to-all comparison we are trying to avoid. We instead read them from the backbone's own features and teach it to produce good ones with a contrastive loss supervised by ground-truth matches at training time only, so at inference the model builds its own sparse attention with no geometry and nothing added to the forward pass. On ScanNet this reaches 55.1 *pose AUC@30 fully self-supervised, within three points of a geometry oracle and ahead of the full-attention baseline, while our indexed kernel runs* 32× *faster than dense FlashAttention at* 96 *frames and keeps scaling to long sequences.*

1. Introduction

Feed-forward models have changed what 3D reconstruction looks like. Where a classical pipeline ran feature matching, geometric verification, and bundle adjustment as separate optimization stages, methods in the DUSt3R line [7, 14] and, most recently, VGGT [13] take a set of images and predict cameras and dense geometry in a single forward pass, with no optimization loop at test time. VGGT does this with a transformer that alternates two kinds of attention: frame attention, where patches attend within their own image, and global attention, where every patch attends to every other patch across all frames. The global blocks are what let the model reason about a scene jointly, and they are also where the cost lives. With S frames and N patches each, global attention is quadratic in SN, so doubling the number of

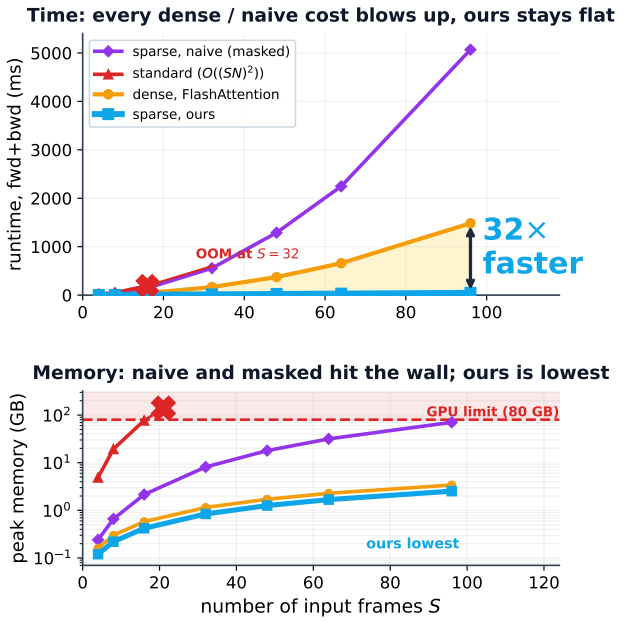

Figure 1. **Sparse cross-view attention lets VGGT scale to long sequences** (one global block, fwd+bwd, A100-80GB, measured). *Top:* standard attention ($O((SN)^2)$) and the masked-sparse kernel, and even FlashAttention, all grow quadratically in time, while ours stays near-linear, 32× faster than FlashAttention at 96 frames and widening. *Bottom:* standard and masked attention are quadratic in memory and run out of memory; FlashAttention and ours are linear, with ours the lowest. FlashAttention fixed dense attention's memory but not its time; ours fixes both. The correspondences are read from features the backbone already computes, so this is gained at the same accuracy, with no ground truth at inference.

views roughly quadruples the work. At VGGT's resolution a single frame already carries $N = 1369$ patches, and the global attention quickly dominates the runtime as S grows. In practice this caps how many views the model can take at once: dense attention gets quadratically slower with every added frame (Fig. 1), so long sequences become too expensive to run, which is exactly the regime where more views would help reconstruction most.

That cost buys less than it looks. A patch has almost nothing to learn from the thousands of patches in other frames that show unrelated parts of the scene; the only informative ones are the few that look at the *same 3D point*. Restricting global attention to those correspondences leaves the answer essentially unchanged, and in fact it is not even a compromise: given oracle correspondences, sparse attention is *better* than dense in our setting (59.1 vs. 50.2 pose AUC@30, Tab. 1), removing distracting cross-talk rather than useful signal. The savings also compound with sequence length: dense attention costs $O((SN)^2)$ while a fixed budget of K correspondences costs $O(SNK)$, so where dense grows quadratically our kernel stays near-linear, already $32\times$ faster at 96 frames (Tab. 2). And it is essentially free: the correspondences are read from features the backbone already computes, so nothing is added at inference.

The difficulty is circular, and it is the real subject of this report: building the sparse index needs correspondences, but finding correspondences across views is the all-to-all comparison we are trying to avoid. The question is therefore not how to exploit a sparse index, but where it comes from cheaply and how accurate it must be.

We get the correspondences from the backbone itself. DINO [9] patch features already match across views, and we teach them to match well: a contrastive loss supervises the embeddings with ground-truth correspondences at training time, so at inference the model finds its own matches with no geometry and nothing added to the forward pass. This recovers most of the oracle's quality, reaching 55.1 pose AUC@30, within 3 points of it and 5 ahead of the full-attention baseline. And these matches need not be precise: corrupting the oracle index, we find the model remarkably tolerant, with half the matches randomly broken still matching dense attention (Sec. 4.3).

Contributions.

- An indexed sparse-attention kernel for VGGT's global blocks that scales near-linearly in the number of frames, $32\times$ faster than dense FlashAttention at 96 frames and widening (Sec. 4.2).
- A noise study of how accurate correspondences must be: half-corrupted matches still match dense attention (Sec. 4.3).
- A contrastive correspondence loss that teaches the backbone to produce its own matches, reaching 55.1 pose AUC@30 with no geometry at inference, within 3 points of the oracle and 5 ahead of the baseline (Sec. 4.5).

The broader takeaway is that when the sparsity pattern is itself a learnable correspondence problem, it pays to learn it with direct supervision rather than to read it off frozen features and hope.

2. Related Work

Feed-forward 3D reconstruction. DUSt3R [14] reframed two-view reconstruction as direct regression of aligned pointmaps, and MASt3R [7] added a matching head for sharper correspondence. Scaling this idea to many views is the active frontier: Fast3R [15] and Spann3R [12] process longer image sets, and VGGT [13] unifies cameras, depth, and geometry in one transformer that alternates within-frame and global attention over all frames at once. The global attention is what lets these models reason jointly, and it is also their cost center, growing quadratically with the number of views. To our knowledge this line has not asked whether that attention needs to be dense, which is the gap we step into.

Efficient and sparse attention. A large body of work reduces the quadratic cost of attention, either with fixed sparsity patterns such as Longformer [2] and BigBird [16], with hashing as in Reformer [6], or with exact IO-aware kernels like FlashAttention [3]; PyTorch FlexAttention [4] makes block-sparse masks practical. Our kernel builds directly on these ideas. The difference is where the sparsity comes from: these methods fix the pattern in advance or derive it from token content within a single sequence, whereas our pattern is defined by 3D geometry across frames and is itself something the model has to predict.

Learned correspondence and matching. Finding which pixels match across views is classic, from learned matchers like SuperGlue [10] and LightGlue [8] to dense methods such as LoFTR [11] and RoMa [5]. Two properties make them an awkward fit here. First, many compute their matches with their own attention over the images, so building our index with one would reintroduce exactly the all-to-all cost we are trying to remove. Second, they operate on image pairs rather than jointly over a set, so the work grows with the number of pairs and scales poorly as views are added. We instead read correspondences from features the backbone already computes: self-supervised descriptors [1, 9] correspond across views for free and over all frames at once.

3. Method

We first recall how VGGT spends its compute on global attention (Sec. 3.1), then define the sparse attention that replaces it (Sec. 3.2; Fig. 2 gives the overview). The rest of

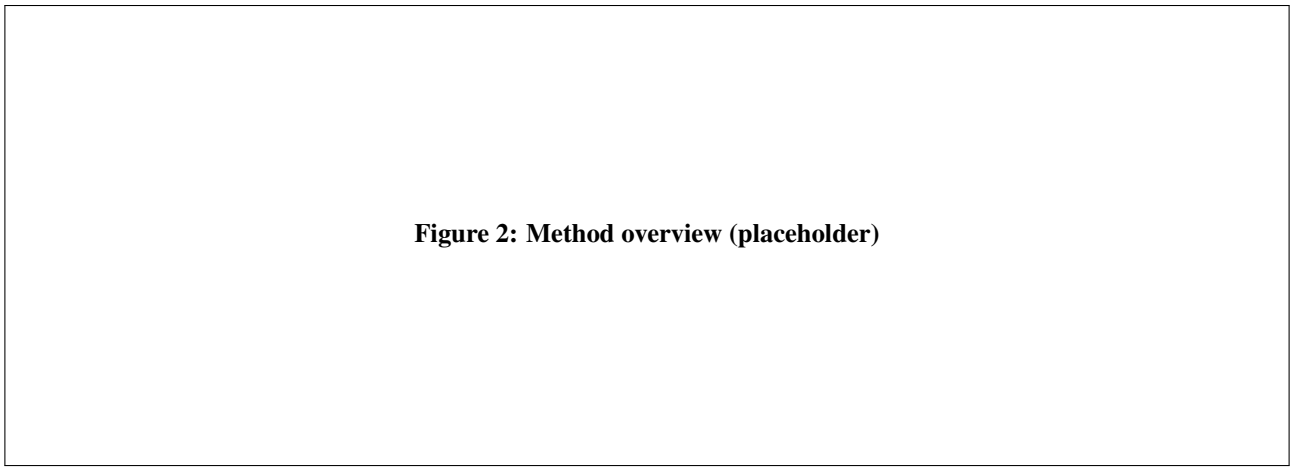

Figure 2. **Method overview.** The sparse index is read off the backbone features the model already computes (cosine similarity + mutual nearest neighbours), and replaces the dense global attention in the aggregator; frame attention stays dense. The contrastive matching loss and its ground-truth correspondences are used only at training time, so at inference the index is built from features alone with no geometry.

the section is about the question that actually matters: where the sparse index comes from (Sec. 3.3) and how we teach the backbone to produce it (Sec. 3.4).

3.1. Background: VGGT and global attention

VGGT [13] takes S images and predicts cameras and dense geometry in one forward pass. Each frame is split into N patches and embedded by a DINO [9] backbone, giving $S\,N$ tokens in total. The aggregator then alternates two attention types: *frame* attention, where the N tokens of one image attend among themselves, and *global* attention, where all $S\,N$ tokens attend to one another so the model can fuse information across views. Camera and depth heads read out the result. Frame attention is linear in S, but global attention compares every token to every other token and so costs $O((SN)^2)$. At VGGT's resolution $N = 1369$, so for even a modest number of views the global blocks dominate memory and runtime and eventually exceed the GPU (Fig. 1). Global attention is therefore the only part we change; everything else in VGGT is left untouched.

3.2. Sparse cross-view attention

The object that drives everything is a *sparse index*: for each query token it lists the few key tokens it is allowed to attend to. Concretely we store an integer tensor of shape $(S\,N) \times K$, where row i holds up to K key indices for query i and unused slots are padded. Standard global attention is the special case where every row lists all $S\,N$ keys; we instead populate each row with the handful of tokens that correspond to the same 3D point as the query, plus the query itself so no token is ever left without a key. Given this index, the global blocks compute a softmax only over the listed keys rather than over all tokens, which is what turns the $O((SN)^2)$ cost into $O(SNK)$ with $K \ll S\,N$. Only the global blocks are made sparse; frame attention stays dense.

Kernel. Realising this saving needs a kernel that never forms the dense scores. We use an indexed FlashAttention kernel. In the forward pass one program handles a query: it walks that query's K key indices, gathers the corresponding K and V rows, and runs the Flash online softmax over them, so the only state kept is the running maximum, denominator, and output, never an $(SN)\times(SN)$ matrix. Each row is padded to a fixed $K_{\max}$ with a negative sentinel that the kernel skips, which keeps the launch shapes static without doing work for empty slots.

The backward is where a sparse pattern is awkward, because a key can be attended by any number of queries, so its gradient is a sum over an irregular set. We split it in two. The query gradient is computed query-parallel, one program per query summing over its own keys, so each query owns its output and there is no contention. The key and value gradients are computed key-parallel using the *transpose* of the index, which for each key lists the queries that attend it; one program per key then walks that list, recomputes the attention weights from the saved softmax statistics, and accumulates the key and value gradients in registers before writing them once. This avoids both the atomic scatter and the two full-size floating-point gradient buffers that a naive sparse backward needs, which is why the kernel is lower in memory than dense attention rather than merely competitive, while staying near-linear in the number of frames (Tab. 2).

3.3. Building the index from backbone features

At test time the index has to come from the model alone. We take the backbone's per-frame patch features, L2-normalise

	Pose AUC ↑				Median err. ↓		Depth ↓		MVS ↓
Method	@5	@10	@20	@30	R°	T°	abs_rel	RMSE	Chamfer
VGGT (baseline)	7.75	20.72	38.68	50.22	**2.07**	12.32	0.0384	0.1307	**3.328**
Geometry-based index (uses depth + poses)									
GeoSparse†	**10.32**	**26.38**	**47.10**	**59.13**	2.13	**9.26**	0.0361	0.1256	3.384
GeoSparse-75	8.00	21.63	40.43	52.44	2.08	11.56	0.0371	0.1265	3.359
GeoSparse-50	5.65	18.28	37.92	50.07	3.13	11.91	**0.0360**	**0.1242**	3.392
Learned index, self-supervised (no geometry at inference)									
EmbSparse	3.51	12.09	27.31	38.45	3.38	18.38	0.0436	0.1533	**3.340**
EmbSparse-FT	9.70	23.51	41.95	53.22	2.27	11.05	**0.0350**	**0.1236**	3.353
MatchSparse	**9.92**	**24.16**	**43.33**	**55.11**	**2.00**	**10.63**	0.0388	0.1278	**3.340**

Table 1. **Reconstruction on ScanNet val** (312 scenes, 10 frames). Each method is evaluated in its native setting: VGGT is the unmodified full-attention baseline, the geometry methods use the index built from depth and poses, and the learned methods build it from their own DINO features with no geometry at test time. Pose AUC and median rotation/translation are camera metrics; depth and MVS Chamfer are geometry metrics. †GeoSparse is the geometry oracle (upper bound, uses ground truth at inference). Per column, **best** and **second best**.

them, and for a pair of frames score every patch against every patch by cosine similarity. We keep a match between two patches only when each is the other's nearest neighbour (mutual nearest neighbours, MNN), which discards the many-to-one matches that dominate raw top-1 and keeps the confident third of patches. This costs nothing beyond a similarity between features we already have, and it produces matches jointly over all frames rather than pair by pair. On their own these matches are only moderately accurate, which is what the next section sets out to fix.

3.4. Teaching the backbone to match

Off-the-shelf features match only moderately well, and the obvious fix, letting the task loss finetune the backbone end to end, cannot improve them. The index is built by a hard selection (an `argmax` followed by the MNN test) that is not differentiable, so no gradient flows through *which* keys are chosen; the backbone receives gradient only through the features the aggregator consumes, not through the matching. Training end to end therefore sharpens the representation but leaves the correspondences essentially unchanged (Sec. 4.6).

Ground-truth correspondences. To supervise the matching we need correct matches to compare against, which we obtain from geometry. We unproject each patch to a 3D point with the ground-truth depth and intrinsics, reproject it into the other frame, and keep the match when the reprojected depth agrees with that frame's own depth within a relative tolerance (5%). This needs depth and poses, so it is available only at training time. The same matches, used directly as the index instead of as supervision, also define the geometry oracle we compare against later (GeoSparse, Tab. 1): an upper bound on what any correspondence source could achieve, not a method one could deploy.

Contrastive matching loss. We give the matching its own gradient with a contrastive loss on the patch features, using those ground-truth correspondences as positives. For a query patch p in frame i with ground-truth match $q^\star$ in frame j, let $f_i(p)$ be its L2-normalised feature; the loss is the InfoNCE objective

$$\mathcal{L}_{\text{match}} = -\log \frac{\exp\big(\langle f_i(p), f_j(q^\star)\rangle/\tau\big)}{\sum_q \exp\big(\langle f_i(p), f_j(q)\rangle/\tau\big)}, \qquad (1)$$

which pulls the features of corresponding patches together and pushes a query away from every other patch in the other frame, with temperature τ and a subsample of pairs per batch for efficiency. The total objective is the usual VGGT camera and depth losses plus $\lambda\,\mathcal{L}_{\text{match}}$. Crucially the ground truth enters only here, as a *training* signal: at inference the model builds its index from its own features with no geometry, exactly as in Sec. 3.3. This is the configuration we call our full method, and it nearly doubles correspondence quality while recovering most of the oracle's reconstruction accuracy (Sec. 4.5).

4. Experiments

Our experiments follow the questions the method raised. We first ask whether sparsity is worth pursuing at all, by measuring the kernel (Sec. 4.2), then how accurate the correspondences must be, by corrupting the oracle index (Sec. 4.3), and how good the free correspondences from a frozen backbone already are (Sec. 4.4). With those bounds in place we report the main reconstruction comparison (Sec. 4.5), and finally show that the contrastive loss is what turns the backbone's mediocre matches into good ones (Sec. 4.6).

4.1. Setup

We evaluate on the ScanNet validation split: 312 scenes of 10 frames for reconstruction, 200 scenes of 8 frames for the correspondence study. We report pose AUC at $5/10/20/30^\circ$ with median rotation and translation error; depth abs_rel, RMSE, and the $\delta < 1.25$ and $\delta < 1.10$ inlier ratios; and the accuracy, completeness, and Chamfer distance of the fused point cloud. Correspondences are scored by precision, recall, and F1 against the geometry oracle at ± 1 patch tolerance. All models train from the public VGGT-1B checkpoint for three epochs, with ground-truth poses rebased to a camera-0 frame and scale-normalised, on A100 GPUs.

One protocol detail matters throughout. A trained model can be run two ways at test time: fed the ground-truth geometry index (comparable across models but needs depth and poses), or building its index from its own features (the deployable setting, no ground truth). The two are not interchangeable, so we always state which a number comes from. The main comparison (Tab. 1) runs each model as trained; swapping the index source is isolated in the ablation (Tab. 7).

4.2. Does sparsity pay off? Kernel benchmarks

Table 2 measures one global-attention block, forward and backward, as the number of frames grows. Standard attention ($O((SN)^2)$), which materialises the $(SN)\times(SN)$ scores, is quadratic in both memory and time: it already needs 77 GB at 16 frames and runs out of memory at 32. Imposing sparsity the obvious way, by handing a mask to an attention kernel, does not help, because it still forms and scores every pair: it is in fact the slowest option of all and runs out of memory near 96 frames. FlashAttention removes the materialisation and makes the memory linear, but it computes the same all-to-all attention, so its runtime stays quadratic. Only an index that is read without ever forming the full set of pairs, as ours is, makes both linear.

Ours is the lowest on both axes. Its key-parallel backward accumulates the key and value gradients in registers rather than scattering them, avoiding the large temporary buffers a generic attention backward allocates and using less memory than even FlashAttention. And because ours scales with the kept keys rather than comparing every token to every other, it is $32\times$ faster than FlashAttention at 96 frames, with the gap widening on every added frame, and runs at sequence lengths the naive cost cannot reach at all. This is what lets the pipeline run on long sequences where dense attention becomes too slow, the prize the rest of the paper earns without ground truth.

The kernel timings leave out one cost: building the index, an all-pairs similarity that is itself $O((SN)^2)$. Table 3 accounts for it end to end, building the index once and reusing it across all L=24 global blocks. As a single batched similarity and mutual-nearest-neighbour pass rather than a full attention, it is a small share of the total (9 ms at 16 frames, 237 of 1365 ms at 96), so the pipeline stays $26\times$ faster than dense at 96 frames; the crossover where sparsity starts to pay is around 8 frames. The build also has its own transient memory peak, from the similarity matrices, which past about 48 frames slightly exceeds the attention peak; even then the pipeline stays below dense FlashAttention (3.1 vs. 3.4 GB at 96 frames), and as a one-time spike before the L blocks it is dwarfed by the activations they store for the backward pass.

	Runtime (ms) ↓		Peak mem. (GB) ↓		runs
Method	S=16	S=96	S=16	S=96	up to
standard ($O((SN)^2)$)	193	**OOM**	**77**	**OOM**	∼16
sparse, naive (masked)	144	5068	2.1	70	∼96
dense, FlashAttention	**42**	**1483**	**0.58**	**3.4**	all
sparse, ours	**8**	**47**	**0.42**	**2.5**	all

Table 2. **Sparse cross-view attention, forward+backward on an A100-80GB** (H=16, D=64, 8 real keys per query padded to $K_{\max}$=128), at two sequence lengths. "runs up to" is the largest S that fits in memory. Our indexed kernel is lowest on every column, $32\times$ faster than FlashAttention at 96 frames and using less memory, and keeps running where the naive costs run out. Per column, **best** and **second**.

4.3. How accurate must correspondences be?

Before chasing a cheap correspondence source, we ask how accurate the index even has to be. We take the oracle index and corrupt it with three operations at controlled rates: *drop* removes a true key, *swap* repoints a key to a random token, and *jitter* moves a key by one patch. We summarise a corruption level by the fraction of matches left intact, $(1-p_{\text{drop}})(1-p_{\text{swap}})(1-p_{\text{jitter}})$, and train a model at each level. The geometry rows of Tab. 1 report two points on this sweep: at 75% intact the model reaches 52.4 AUC@30, close to the clean oracle at 59.1, and at 50% intact it still reaches 50.1, matching the full-attention baseline (VGGT, 50.2). In other words, the model tolerates a remarkably broken index, which is the result that makes a cheap, imperfect source viable in the first place.

4.4. How good are the free correspondences?

Before training the backbone we measure how good its matches already are. Taking the most similar patch in the other frame (raw top-1) reaches 47.9% recall at 24.0% precision, since it forces every patch to match something; mutual nearest neighbours instead keep the confident third of patches, raising precision to 41.1% at 27.3% recall. The errors that remain are mostly near-misses: the large majority land within a patch or two of the correct location and only about 0.7% are gross failures more than ten patches away.

	ours runtime (ms) ↓			vs	peak memory (MB) ↓		
S	index	attn $\times L$	total	dense	index	attn	pipeline
4	2	72	74	1.0×	109	112	114
8	3	96	99	2.7×	213	219	225
16	9	192	201	**5.0**×	413	422	433
32	27	384	411	**9.7**×	824	842	864
48	60	576	636	**14.1**×	1304	1255	1304
64	105	744	849	**18.6**×	1863	1682	1863
96	237	1128	1365	**26.1**×	3143	2533	3143

Table 3. **End-to-end pipeline, with the index cost included** (L=24 global blocks, fwd+bwd, A100-80GB). *Time:* the index is built once (*index*) and reused by the sparse attention in every block (*attn*); *total* is their sum and the next column its speedup over dense FlashAttention in all L blocks. *Memory:* transient peak of the build, of one attention block, and of the two combined (*pipeline*, their max). The build peak overtakes the attention peak past ∼48 frames but the pipeline (3.1 GB at 96) stays below dense FlashAttention's 3.4 GB. **best** / **second** speedup.

Filter on MNN (EmbSparse)	Precision	Recall	F1
MNN (no filter)	41.1	27.3	**32.8**
+ cosine sim > 0.9	*no effect (> 0.7 already)*		
+ Lowe ratio < 0.9	0.0	0.0	0.0
+ displacement margin	56.3	3.8	**7.1**

Table 4. **Similarity-based filtering does not help** (EmbSparse matcher, ±1 patch). No threshold on cosine similarity, nearest-neighbour ratio, or displacement margin improves F1 over plain MNN. Best and second F1 in **green** / **blue**.

This is also why we score correspondences at a strict ±1 patch tolerance: at the resolution of these tokens a ±5 patch "hit" already spans roughly a third of the image, so a looser threshold would flatter every method.

A natural hope is that a confidence threshold could clean up the matches, but it cannot (Tab. 4). A cosine cutoff removes nothing, because every mutual match already scores above it, and ratio or displacement filters raise precision only by collapsing recall to near zero. Good and bad matches are not separable by confidence, so the features themselves have to change. The frozen backbone is, in short, usable but mediocre, which is what motivates training it to match. Further correspondence-quality analysis (miss-distance distribution, patch-radius reference, full tolerance table) is given in Appendix 6.

4.5. Main results: reconstruction

Table 1 compares all models, each run as it was trained. The first thing to note is that sparsity is not a compromise: given good correspondences, the geometry oracle GeoSparse reaches 59.1 pose AUC@30 against 50.2 for the full-attention VGGT baseline, so removing the irrelevant cross-view attention actually helps. The question is then how close a method with no geometry at test time can get to that oracle.

Free correspondences alone do not get there: EmbSparse, which builds its index from the frozen backbone, reaches only 38.5, well below the baseline. Training the backbone changes the picture. EmbSparse-FT already passes the full-attention baseline at 53.2, and adding the matching loss, MatchSparse, reaches 55.1, within about three points of the oracle and five ahead of VGGT, with the best median rotation error of any trained model (2.0°). All of this is self-supervised, using no geometry at inference. The one cost is a small depth regression on the MatchSparse row, which trades against matching quality and is controllable through the loss weight λ.

Every row here is evaluated in its native setting, so this is an honest comparison between methods rather than between inference tricks. What happens if we instead hand the learned models a clean geometry index at test time is isolated in the ablation (Tab. 7).

4.6. Does the contrastive loss fix the matches?

The reconstruction gains trace back to the matches themselves (Tab. 5). Finetuning the backbone end to end moves correspondence F1 by only 1.1 points, from 32.8 to 33.9, exactly as the non-differentiable index predicts: the task loss never reaches the matching. The contrastive loss, which does reach it, nearly doubles F1 to 58.4, with precision and recall both jumping by about 25 and 23 points. The improvement is not an artifact of the tolerance we chose: it holds at every threshold from ±0 to ±10 and is largest at the strict ±1 and ±0 that pose accuracy depends on (Tab. 6). This is the direct evidence that giving the backbone a gradient for matching, rather than hoping finetuning would supply one, was the real bottleneck.

The same story is visible directly in the matches (Fig. 3). On the wide-baseline pairs that are hardest, the frozen backbone produces a tangle of mostly wrong correspondences, and finetuning leaves it largely unchanged; only after the matching loss do the links become clean and mostly correct.

4.7. Ablations

Index source at inference. A learned model can be fed either its own embedding index or the clean ground-truth index at test time, and which one it prefers is revealing (Tab. 7). EmbSparse-FT does slightly *better* with its own imperfect matches than with the geometry index (53.2 vs. 52.3 AUC@30): having been trained only against its own selections, it has adapted to them. MatchSparse reverses this and gains from the clean index (56.5 vs. 55.1),

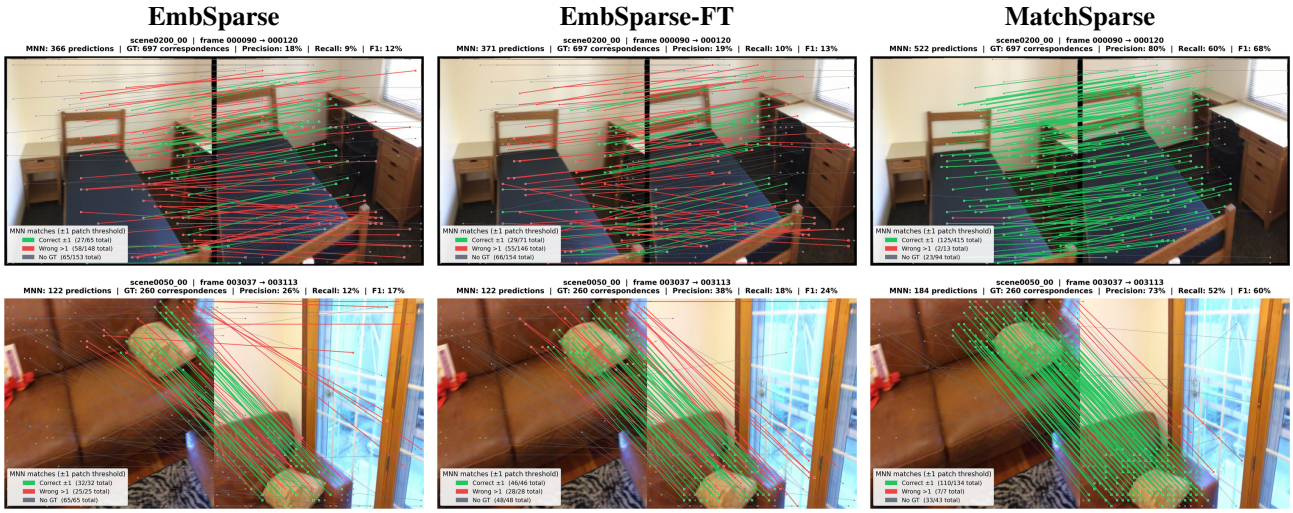

Figure 3. **Correspondences across the three backbones** on two ScanNet frame pairs (rows), drawn for the same query patches. Lines are mutual nearest neighbour matches: green is correct within ±1 patch, red is wrong, grey is unverifiable (no ground-truth depth). The per-panel precision, recall, and F1 are printed in each image. The frozen backbone matches poorly and finetuning barely changes it, while MatchSparse turns the same pairs mostly green, exactly the jump in Tab. 5.

Method	Precision ↑	Recall ↑	F1 ↑
EmbSparse	41.1	27.3	32.8
EmbSparse-FT	**43.4**	**27.8**	**33.9**
MatchSparse	**68.3**	**51.0**	**58.4**

Table 5. **Correspondence quality on ScanNet val** (200 scenes, MNN matches at ±1 patch). Each row is the correspondences produced by that method's backbone. Per column, **best** and **second best**.

	F1 at tolerance (patches) ↑					
Method	±0	±1	±2	±3	±5	±10
EmbSparse	15.5	32.8	38.7	42.3	46.6	51.4
EmbSparse-FT	**15.6**	**33.9**	**40.0**	**43.5**	**47.5**	**51.7**
MatchSparse	**32.4**	**58.4**	**61.8**	**63.0**	**64.2**	**65.3**

Table 6. **Correspondence F1 vs. match tolerance** (MNN, 200 scenes). A match is correct if within the given patch distance of the ground-truth correspondence. Per column, **best** and **second best**.

which says the matching loss pushed its features toward geometry-consistent correspondences rather than merely self-consistent ones. This is the bridge between "the backbone learns to match" and "the matches are actually geometric."

	AUC@30 ↑		Median R° ↓	
Method	own emb.	GT idx.	own emb.	GT idx.
EmbSparse	38.45	**39.01**	**3.38**	3.42
EmbSparse-FT	**53.22**	52.29	**2.27**	2.78
MatchSparse	55.11	**56.54**	**2.00**	2.13

Table 7. **Index source at inference.** For each learned model, the sparse index built from its own embeddings (deployable, as in Tab. 1) vs. the ground-truth geometry index. Per row, **better** of the two sources for each metric.

5. Conclusion

VGGT's cross-view global attention can be made sparse along correspondences without losing accuracy, and the correspondences can come from the model's own backbone rather than geometry. A training-time contrastive matching loss teaches that backbone to produce its own matches, nearly doubling correspondence quality and reaching 55.1 pose AUC@30, within three points of the geometry oracle and ahead of the full-attention baseline, with no ground truth at inference.

The transferable lesson is that when the sparsity pattern is itself a correspondence problem, it is better to learn it with direct supervision than to read it off frozen features and hope. That the index selection blocks the gradient, so finetuning alone cannot improve the matches, is the part most likely to carry over to other models that route computation through a discrete choice.

Limitations. The learned model is still about three points short of the oracle, and the matching loss trades a little depth accuracy for better correspondences. Our study is on ScanNet indoor scenes, and the kernel's near-linear scaling is measured on a single global block; the end-to-end win on a full long-sequence reconstruction is not yet demonstrated.

What is next. The most immediate step is to realise the scaling claim in practice: measure peak memory and runtime end to end as frames grow, and reconstruct the long sequences dense attention cannot fit (Fig. 1), where sparse attention should pay off most. The method should then *scale* along two axes: breadth of data, training and evaluating the learned matcher beyond ScanNet on outdoor, object-centric, and in-the-wild captures to test whether its correspondences transfer; and stronger backbones, applying the same recipe to newer, larger models such as VGGT-Ω, where global attention's quadratic cost bites harder and sparsity should save more. We also expect the gap to the oracle to close with a longer schedule and a tuned loss weight, and leave finishing the kernel-time end-to-end win (the FlexAttention path) as ongoing work.

References

[1] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep ViT features as dense visual descriptors. In *ECCVW*, 2022. 2

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020. 2

[3] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022. 2

[4] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. FlexAttention: The flexibility of PyTorch with the performance of FlashAttention, 2024. PyTorch. 2

[5] Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. RoMa: Robust dense feature matching. In *CVPR*, 2024. 2

[6] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020. 2

[7] Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding image matching in 3d with MASt3R. In *ECCV*, 2024. 1, 2

[8] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. LightGlue: Local feature matching at light speed. In *ICCV*, 2023. 2

[9] Maxime Oquab, Timothée Darcet, Théo Moutakanni, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024. 2, 3

[10] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 2

[11] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. LoFTR: Detector-free local feature matching with transformers. In *CVPR*, 2021. 2

[12] Hengyi Wang and Lourdes Agapito. 3D reconstruction with spatial memory. In *3DV*, 2025. 2

[13] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *CVPR*, 2025. 1, 2, 3

[14] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jérôme Revaud. DUSt3R: Geometric 3d vision made easy. In *CVPR*, 2024. 1, 2

[15] Jianing Yang, Alexander Sax, Kevin J. Liang, Mikael Henaff, Hao Tang, Ang Cao, Joyce Chai, Meta, et al. Fast3R: Towards 3d reconstruction of 1000+ images in one forward pass. In *CVPR*, 2025. 2

[16] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020. 2

(Full) Attention is More Than You Need: Sparse Cross-View Attention in VGGT

Supplementary Material

6. Correspondence quality: additional analysis

This appendix collects the extra correspondence-quality material referenced from Sec. 4.4, which did not fit in the main paper.

Miss-distance distribution. The errors a frozen backbone makes are mostly near-misses rather than gross failures. Fig. 4 plots the distance, in patches, between each predicted match and its ground-truth token. For mutual nearest neighbours 60.7% of matches land within ± 1 patch and the distribution falls off sharply: only about 4.9% are farther than ten patches and 0.7% farther than twenty. Raw top-1 has a heavier tail because it forces every patch to match something, which is exactly what the mutual-nearest-neighbour test removes.

Patch-radius reference. Because we score correspondences at a strict ± 1 patch tolerance, it helps to visualise what a given patch radius covers on the image (Fig. 5). At the resolution of these tokens a ± 5 patch "hit" already spans roughly a third of the image, which is why a looser threshold would flatter every method.

Full tolerance numbers. The complete precision/recall/F1 of all three matchers across tolerances from ± 0 to ± 10 patches is reported in Tab. 6 of the main paper.

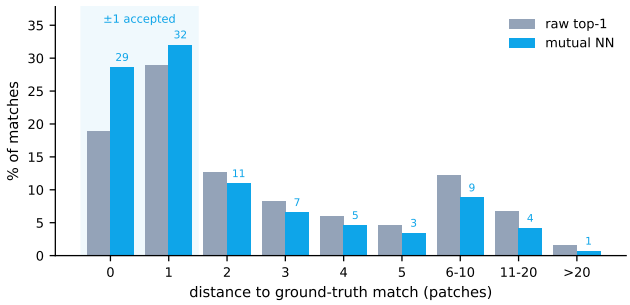

Figure 4. **Miss-distance distribution** on the ScanNet correspondence split (200 scenes, 8 frames each). Fraction of matches at each distance from the ground-truth token, for raw top-1 and mutual nearest neighbours. The shaded band is the ± 1 tolerance the paper scores at; both matchers concentrate there and gross failures beyond ten patches are rare.

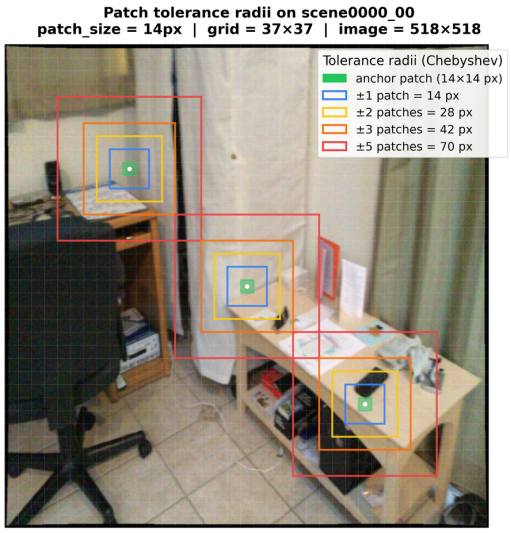

Figure 5. **Patch-radius reference.** Concentric tolerance radii drawn around a query patch on a real frame. A ± 1 patch radius is a tight neighbourhood, while ± 5 already covers a large fraction of the image, motivating the strict threshold used throughout.