RESEARCH ARTICLE

LoGA: Loop-Carried Graph Attention Based Looped Vision Transformers for Fine-Grained Plant Disease Classification

Subham Divakar[1,*] **and Rojalina Priyadarshini**[2,*]

[1] Department of Electronics and Telecommunications, C. V. Raman Global University, Bhubaneswar, Odisha 752054, India
[2] Department of Computer Science, C. V. Raman Global University, Bhubaneswar, Odisha 752054, India

Abstract

Fine-grained plant disease classification requires separating classes of the same crop that often differ only in small, sparse lesion patterns. Looped Vision Transformers (LoopViTs) obtain depth by applying a small stack of weight-tied blocks repeatedly, yet each block rebuilds its attention graph from scratch at every iteration, although the same block has just computed a graph over the same tokens. We propose LoGA, a looped vision transformer with *Loop-Carried Graph Attention*, in which the attention graph of each block at iteration t is biased by the graph the *same* block produced at iteration $t-1$, used as a tempered prior. In its *log-prob* form the update $A_t \propto \exp(QK^\top/\sqrt{d}) \odot A_{t-1}^{\gamma}$ is, when the floor is inactive, equivalent to attending with a geometrically discounted sum of past query-key logits; a flash-compatible *qk* variant carries one step of raw logits. LoGA adds 24 scalars to a 9.91 M-parameter model and reduces exactly to vanilla LoopViT at initialisation. We also study a parameter-free graph-stability signal Δ_t and show that, under label smoothing, predictive entropy stays near 0.671 nats, so a common entropy-exit threshold is never reached. Trained from scratch, LoGA-logprob reaches 99.78%, 94.2%, 98.6% and 99.53% top-1 accuracy on the held-out splits of PlantVillage, PlantDoc, Cassava and Plant Pathology 2020, versus 98.11%, 88.3%, 84.4% and 91.7% for vanilla LoopViT. On PlantVillage, the CLS-attention graph contracts across iterations and accuracy is maintained at twice the training depth. We regard these results as encouraging first evidence and discuss their limitations.

Keywords: Fine-grained plant disease classification, looped transformers, vision transformer, attention graph, weight tying, recurrent depth, dynamic exit.

1 Introduction

Plant diseases cause estimated yield losses of 10-16% globally each year [1], which makes fast and accurate visual diagnosis valuable, particularly on low-cost devices used in the field. Vision Transformers

Submitted: 23 September 2026
Revised: 23 October 2026
Accepted: 26 October 2026
Published: 29 October 2026

Vol. 3, **No.** 4, 2026.
10.62762/TETAI.2026.672726

***Corresponding authors:**
✉ Subham Divakar
shubham.divakar@gmail.com
✉ Rojalina Priyadarshini
priyadarshini.rojalina@gmail.com

Citation
Divakar, S., & Priyadarshini, R. (2026). LoGA: Loop-Carried Graph Attention Based Looped Vision Transformers for Fine-Grained Plant Disease Classification. *ICCK Transactions on Emerging Topics in Artificial Intelligence*, 3(4), 218–238.

© 2026 by the Authors. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

(ViTs) [2, 3] have become a common backbone for image recognition, including agricultural tasks such as leaf-disease diagnosis [4, 5]. Their depth, however, is paid for in parameters: a 12-layer ViT-S has roughly 22 M parameters, which can be demanding for edge devices. *Looped* (or recurrent-depth) transformers [6–8] decouple depth from parameter count. A small stack of B weight-tied blocks $\mathcal{M}_\theta$ is applied T times, $z_{t+1} = \mathcal{M}_\theta(z_t + e_t)$, giving an effective depth of $B \cdot T$ with the parameters of only B blocks. Shu *et al.* [9] recently introduced LoopViT for visual abstract reasoning on the Abstraction and Reasoning Corpus (ARC). LoopViT combines weight-tied hybrid attention-convolution blocks with an entropy-based dynamic exit and observes that predictions "crystallise" over iterations as predictive entropy falls. In this work we adapt the LoopViT recurrent core to image classification, with 2-D rotary position embeddings and a CLS-token readout, and use it as the backbone of LoGA. The use of looped transformers for fine-grained visual recognition in resource-constrained agricultural settings represents an emerging and largely unexplored direction.

Our starting point is a simple observation. In a looped transformer, block b at iteration t and block b at iteration $t+1$ are *the same function*. Yet at every iteration each block builds its attention graph $A_t^b = \mathrm{softmax}(Q_t^b K_t^{b\top}/\sqrt{d})$ from scratch, without access to the graph it computed one iteration earlier. This graph is a $T_{\mathrm{tok}} \times T_{\mathrm{tok}}$ row-stochastic matrix encoding which patches each patch attends to. Recomputing it independently may leave useful information unused, which could matter for *fine-grained* recognition. In leaf-disease diagnosis, classes of the same crop (e.g. tomato early blight versus late blight versus target spot) can differ only in the shape, colour and spatial distribution of a few lesion patches. A model may need to route attention repeatedly between these sparse lesion patches and the rest of the leaf, and the routing found at one iteration is not directly available at the next.

Carrying or reusing attention across depth is not new in itself. RealFormer [10] adds the previous layer's attention scores to the current layer's, Evolving Attention [11] convolves over them, and attention-sharing methods [12, 13] reuse the maps of earlier layers to save computation. In a standard (untied) transformer, however, layer l and layer $l+1$ have different weights and roles, so their graphs are not estimates of the same quantity. In a looped transformer the situation is different: carrying the graph of block b to block b turns the loop into an *iterative estimator of one graph per block*. Loop-Carried Graph Attention (LoGA) is built on this property.

Contributions.

1. **LoGA architecture and mechanism.** We introduce LoGA, a looped vision transformer for fine-grained plant disease classification whose attention uses a loop-carried attention prior. It has two instantiations: a *log-prob* form, which carries the normalised log-attention graph recursively, and a flash-compatible *qk* form, which carries one step of raw logits through an exact head-dimension concatenation identity. LoGA adds $B \times h$ scalars (24 in our configuration, about 0.00024% of the model) and is exactly vanilla LoopViT at initialisation.

2. **Analysis.** We show that log-prob LoGA is equivalent to attending with a geometrically discounted sum of past logits when the floor is inactive (Proposition 1), that the floor c bounds the prior's influence on any attention ratio (Proposition 2), and that the qk form is exact (Proposition 3).

3. **Graph-stability signal and a note on entropy exits.** We define a parameter-free signal Δ_t from consecutive CLS-attention rows, and we derive the entropy level implied by label smoothing, which explains why a low entropy-exit threshold is not reached under common training recipes. In our PlantVillage run, the default entropy exit used all 12 block applications for every image in every epoch.

4. **Measured loop dynamics.** On PlantVillage we report the per-iteration accuracy, predictive entropy and graph-change trajectory of LoGA over 98 epochs. The CLS-attention graph first changes strongly between iterations and later contracts across iterations as training converges.

5. **Where the gain comes from.** We separate the effect of the carried prior from that of the extra gradient path through the graph, with a stop-gradient (`carry_detach`) ablation and with test-time interventions that remove the prior or replace it by the graph of a different-class image, and we examine per-iteration class activation maps of LoGA and vanilla LoopViT on the same images, for three visually similar tomato diseases of PlantDoc (bacterial spot, early blight and late blight).

6. **Evaluation on plant-pathology benchmarks.** We

evaluate on four leaf-disease datasets ranging from laboratory images (PlantVillage) to field images (PlantDoc, Cassava, Plant Pathology 2020), with controls for residual attention and parameter count, and place the results alongside published ViT, MobileViT, Swin Transformer [14], Vision Mamba and convolutional neural network (CNN) methods. The code is released as open source.

The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 develops LoGA and the exit mechanism. Section 4 describes the experimental protocol, Section 5 reports and discusses results, and Section 6 concludes.

2 Related Work

2.1 Weight Sharing and Recurrent Depth

Universal Transformers [6] apply one transformer block recurrently with shared weights; Shen *et al.* [15] extend this idea to vision transformers by sharing weights recursively across depth, constructing parameter-efficient deep networks without increasing parameter count. MiniViT [16] multiplexes weights across consecutive vision transformer blocks, achieving up to $9.7\times$ parameter reduction without seriously compromising ImageNet accuracy. Deep equilibrium models [17] take the limit $T \to \infty$ and solve for a fixed point. Looped transformers have been shown to emulate programmable computers [7] and to learn iterative algorithms with better length generalisation [8]. LoopViT [9] applies looped transformers to visual abstract reasoning (ARC), using weight-tied hybrid blocks, step embeddings and entropy-based dynamic exit; we adopt its recurrent core as our backbone. To our knowledge, these works carry the *token state* z_t across iterations but not the *attention graph*.

2.2 Residual, Evolving and Shared Attention

RealFormer [10] adds the pre-softmax scores of layer $l-1$ to those of layer l. Evolving Attention [11] applies convolutions to the previous layer's attention maps before adding them. Xiao *et al.* [12] share attention weights across adjacent layers to speed up inference, and Skip-Attention [13] reuses the self-attention of earlier ViT layers through a lightweight parametric function. These works indicate that attention maps are strongly correlated across depth. LoGA differs from them in three ways. (i) It carries the graph across *iterations of the same weight-tied block*, not across different layers. (ii) Its primary form carries a *normalised* log-probability graph with an explicit floor, which gives a recursive prior-times-likelihood interpretation. (iii) It uses the stability of the carried graph as a candidate halting signal. We include a RealFormer-style control, residual scores across blocks within one iteration, to examine whether gains come mainly from residual attention.

2.3 Rank Collapse and Oversmoothing

Pure self-attention loses rank doubly exponentially with depth [18], and deep ViTs oversmooth patch tokens towards a low-frequency subspace [19]. A sticky attention prior could in principle accelerate this collapse, which is a risk to keep in mind when carrying graphs across iterations. We therefore report Dirichlet energy and effective rank per iteration as diagnostics.

2.4 Adaptive Computation and Early Exit

BranchyNet [20] and Bakhtiarnia *et al.* [21] exit when an intermediate classifier is confident, the latter designing early-exit branches specifically for vision transformers. PonderNet [22] learns a halting distribution, and A-ViT [23] halts individual tokens. Confidence-based exits implicitly assume that predictive entropy can reach small values. We note that label smoothing [24] limits how low it can go, and we consider a complementary criterion based on the internal graph rather than the output.

2.5 Deep Learning for Fine-Grained Plant Disease Classification

PlantVillage [25] is the standard laboratory benchmark. Transfer-learned CNNs reached 99.35% on it a decade ago [4], and a ViT with a mixture-of-experts classifier recently reported 99.96% [26]. At this level of saturation, PlantVillage mainly separates methods in the from-scratch, parameter-constrained regime that we study. PlantDoc [5] is much harder: 2,598 field photographs with real backgrounds, occlusion and multiple leaves per image, on which the same model families obtain 59-74% [26]. Cassava [27, 28] and Plant Pathology 2020 [29] add further field conditions, strong class imbalance, label noise and co-occurring diseases.

Transformer-based plant disease work includes ViTs trained from scratch with fewer than one million parameters [30] and a mixture-of-experts ViT for images in the wild [26]. ViT-SmartAgri [31] couples a standard ViT with a smartphone deployment pipeline and reports 90.99% on the ten tomato classes of PlantVillage, and PMVT [32] designs a lightweight MobileViT variant for mobile devices (93.6% on a

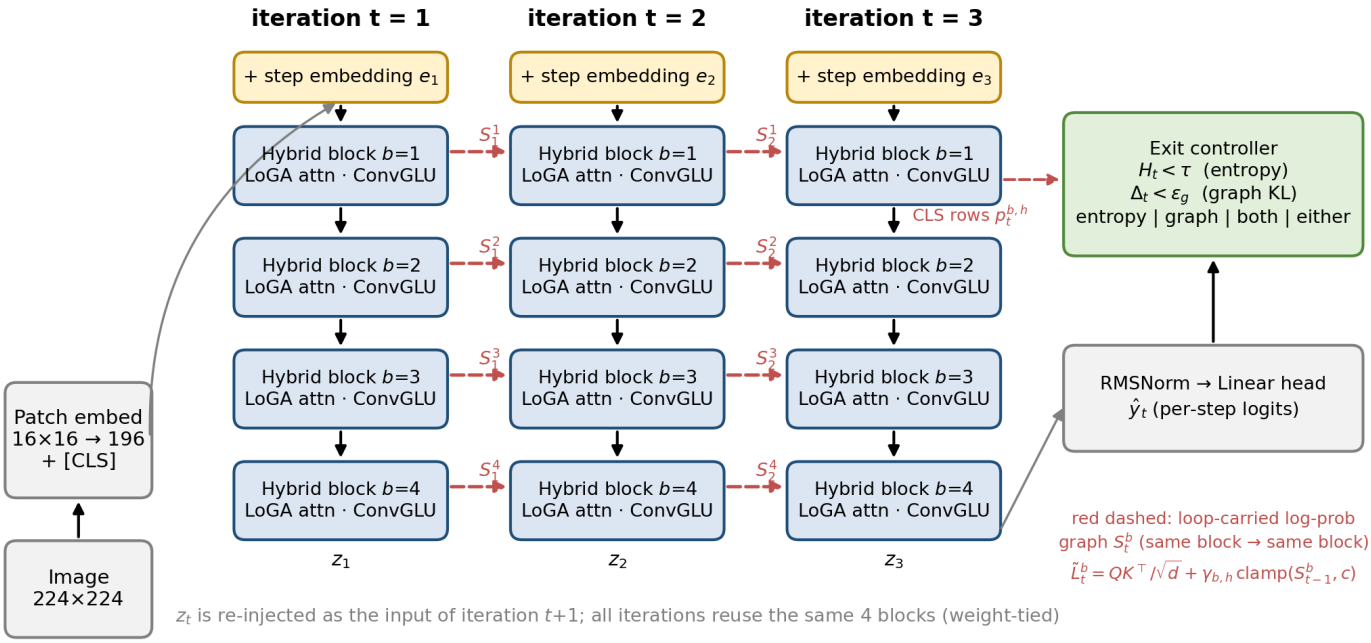

Figure 1. Overview of the LoGA architecture (a looped vision transformer with Loop-Carried Graph Attention). The same four hybrid blocks are applied at every iteration (weight-tied). Each block carries its own attention graph S_t^b to itself at the next iteration (red dashed), where it acts as a tempered prior on the query-key logits. CLS-attention rows feed a graph-stability signal Δ_t which, together with the predictive entropy H_t, can drive per-sample dynamic exit.

wheat disease dataset). These methods use a single backbone with standard, iteration-free attention. For field data, Zhang and Liu [33] apply a multi-scale Vision Mamba to a filtered version of PlantDoc and report 92.67%, the highest published PlantDoc figure we are aware of. On Cassava, Jain and Nain [34] concatenate EfficientNet-B0 and DenseNet-121 features to reach 97.61%, and Saood *et al.* [35] report 96.80% with a customised Swin Transformer. For apple leaves, Kaur and Kaur [36] report 99.42% with PomaNet, an optimised CNN. Several of these methods build on ImageNet-pre-trained backbones or do not state their initialisation; training ViTs on small datasets from scratch remains difficult [37, 38], and this is the setting in which a stronger inductive bias and a parameter-efficient looped backbone may help.

LoGA continues our earlier work on plant disease classification in two specific respects. First, recursive additive attention [39] refined a CNN's spatial attention map by re-applying the same attention module several times within one forward pass; LoGA transfers this idea of iteratively refining one attention estimate from a CNN to a weight-tied looped transformer, where the refined object is the full token-to-token attention graph of each block and the refinement happens across loop iterations. Second, SMOTE-DL [40] addressed the strong class imbalance of leaf-disease data, which is also the reason we report class-balanced metrics for the imbalanced Cassava and Plant Pathology 2020 datasets.

3 Proposed Method

3.1 Overview

Figure 1 shows the complete LoGA architecture, Figure 2 contrasts its weight-tied loop with an untied ViT, Figure 3 details one LoGA attention module, and Figure 4 gives a schematic illustration of the looped graph computation. An image is split into 16×16 patches, embedded, and prefixed with a CLS token. The token sequence then enters a recurrent core of $B = 4$ hybrid blocks that is unrolled for $T = 3$ iterations (12 block applications). Within each block, LoGA attention receives the carried graph state S_{t-1}^b of the same block from the previous iteration (red dashed arrows) and emits an updated state S_t^b. After each iteration, a shared normalisation layer and linear head produce per-step logits $\hat{y}_t$. An exit controller can combine output entropy H_t and graph change Δ_t to decide, per sample, whether to stop.

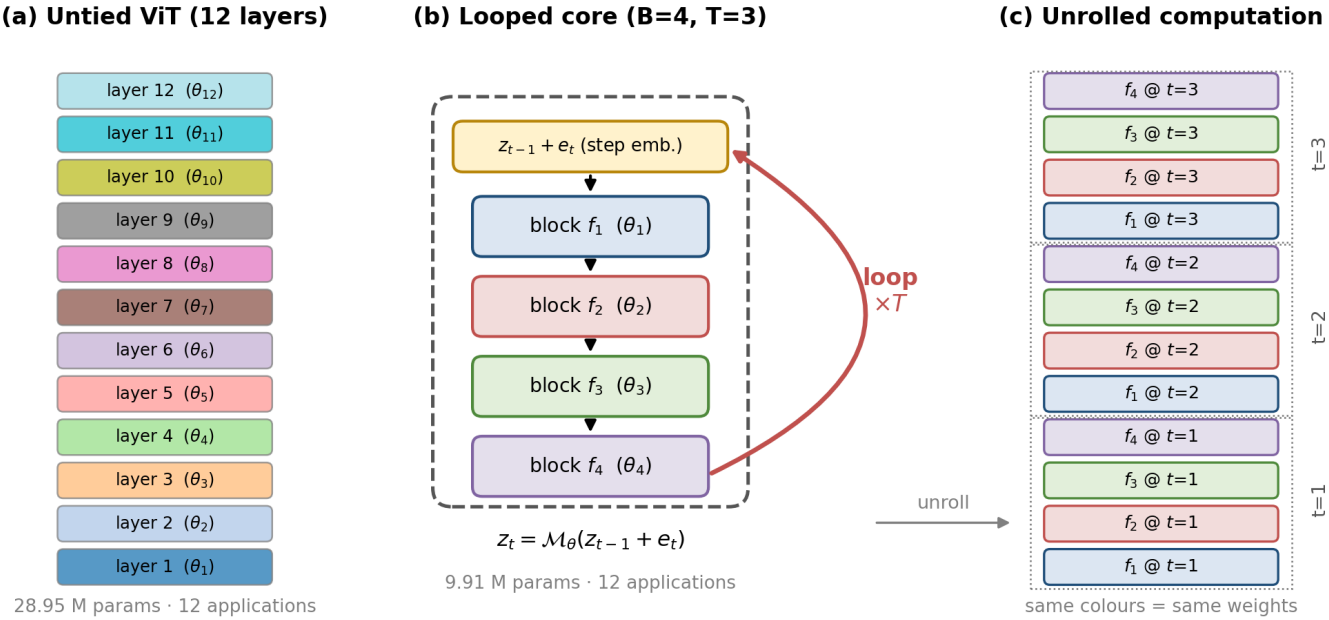

Figure 2. The loop. (a) An untied ViT uses a different set of weights θ_ℓ for each of its 12 layers (28.95 M parameters). (b) The looped core of LoGA holds only $B = 4$ blocks $f_1, \ldots, f_4$; after adding the step embedding e_t, the same four blocks are applied again at every iteration, $z_t = \mathcal{M}_\theta(z_{t-1} + e_t)$ (9.91 M parameters). (c) Unrolled over $T = 3$ iterations, the looped core also performs 12 block applications, but block f_b at iteration t and at iteration $t + 1$ is *the same function*. This is what makes it meaningful to carry the attention graph of f_b from one iteration to the next.

3.2 Notation

Let N denote batch size, h the number of heads, $d = D/h$ the head dimension, P the number of patches and $T_{\text{tok}} = P + n_{\text{cls}}$ the number of tokens ($T_{\text{tok}} = 197$ for a 224×224 image with 16×16 patches and one CLS token). Iterations are indexed $t \in \{1, \ldots, T\}$ and blocks $b \in \{1, \ldots, B\}$. For block b at iteration t, $Q_t^b, K_t^b, V_t^b \in \mathbb{R}^{N \times h \times T_{\text{tok}} \times d}$ are the per-head queries, keys and values after rotary encoding.

3.3 Backbone: Looped Vision Transformer

Patch embedding. A 16×16 convolution with stride 16 maps $x \in \mathbb{R}^{3 \times 224 \times 224}$ to $P = 196$ tokens of width $D = 384$, and a learnable CLS token is prepended: $z_0 \in \mathbb{R}^{T_{\text{tok}} \times D}$.

Recurrent core. The core $\mathcal{M}_\theta = f_B \circ \cdots \circ f_1$ consists of $B = 4$ hybrid blocks applied T times:

$$z_t = \mathcal{M}_\theta(z_{t-1} + e_t), \qquad t = 1, \ldots, T, \tag{1}$$

where $e_t \in \mathbb{R}^D$ is a learnable *step embedding* broadcast over tokens. It lets the shared weights distinguish iterations. For $t > T_{\text{train}}$, the last embedding is reused ($e_t = e_{T_{\text{train}}}$), which enables depth extrapolation at inference.

Hybrid block. Each block uses pre-normalisation with RMSNorm and stochastic depth [41] with rate 0.1, linearly increasing over the $B \cdot T$ unrolled applications:

$$u = z + \text{DropPath}\big(\text{Attn}(\text{RMSNorm}(z))\big), \tag{2}$$

$$z' = u + \text{DropPath}\big(\text{ConvGLU}(\text{RMSNorm}(u))\big). \tag{3}$$

The feed-forward network is a heterogeneous convolutional gated linear unit (GLU) [42]. A linear layer expands $D = 384$ to $2 \cdot 1536$ channels and splits them into a gate g and a value v. The patch tokens of the value branch are reshaped to a 14×14 grid and passed through a 3×3 depth-wise convolution, which gives every block an explicit local inductive bias. The output is $W_2\big(\text{SiLU}(g) \odot \text{DWConv}(v)\big)$ with $W_2 : 1536 \to 384$; the CLS token bypasses the convolution.

2-D rotary position embedding. Queries and keys of patch tokens are rotated by a 2-D rotary position embedding (RoPE) [43, 44]. Half of each head's channels encode the row index and half the column index, so $QK^\top$ depends on relative 2-D displacement. CLS tokens are not rotated. Because $d = 64$ is divisible by 4, the axial split is exact.

Readout. After each iteration, $\hat{y}_t = W_{\text{head}}\,\text{RMSNorm}(z_t)_{\text{cls}}$. Training uses the final-iteration logits $\hat{y}_T$, while $\hat{y}_1, \ldots, \hat{y}_T$ (and $\hat{y}_{T+1}, \ldots, \hat{y}_{2T}$ for extrapolation) are used for analysis and dynamic exit.

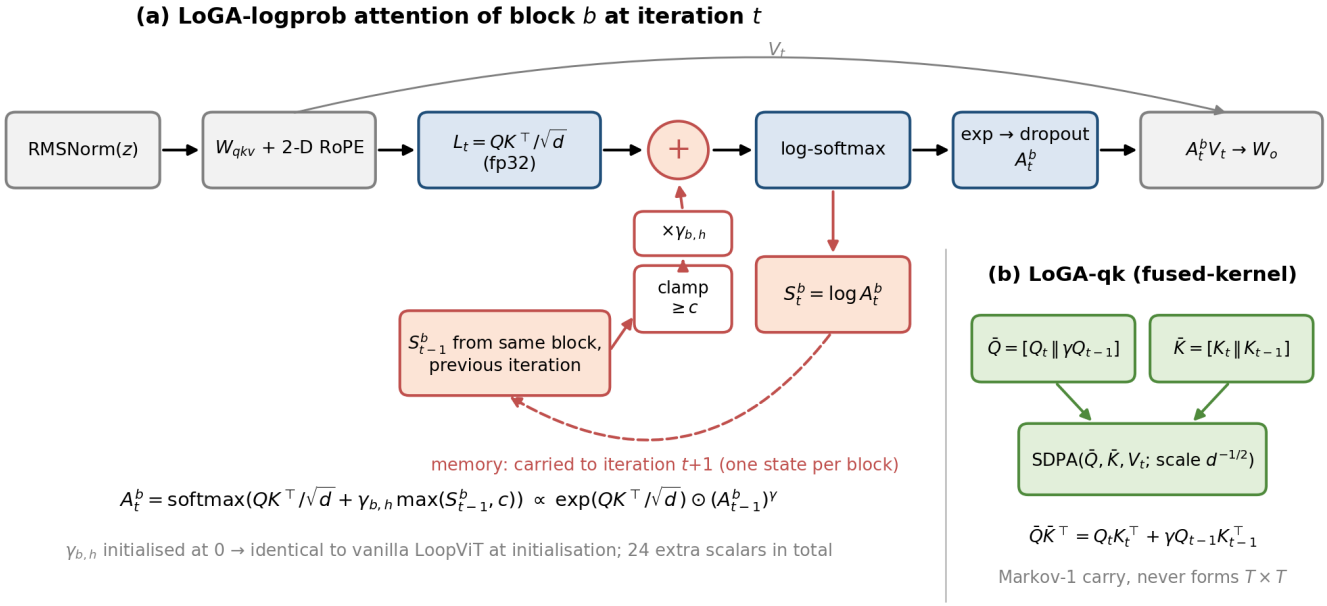

Figure 3. The LoGA attention module. (a) Log-prob carry (primary): the query-key logits L_t of block b are computed in fp32, and the block's own log-attention graph from the previous iteration, S^b_{t-1}, is floored at c, scaled by the learnable per-head gate $\gamma_{b,h}$ and added before the log-softmax (Eq. 6). The new graph $S^b_t = \log A^b_t$ is stored and carried to the next iteration (dashed). (b) QK carry: the previous queries and keys are concatenated along the head dimension so a fused scaled dot-product attention (SDPA) kernel computes Eq. (11) exactly (Proposition 3).

With $D = 384$, $h = 6$ and an MLP ratio of 4, one hybrid block has 2,380,416 parameters. The recurrent core has 9,521,664 and the full model 9,909,158 (9.91 M). An untied network with the same 12 block applications would have 28,952,486 parameters, so looping reduces the parameter count by $2.92\times$ at equal compute.

3.4 Vanilla Attention in a Loop

The attention of block b at iteration t in vanilla LoopViT is

$$L^b_t = \frac{Q^b_t (K^b_t)^\top}{\sqrt{d}}, \qquad A^b_t = \mathrm{softmax}(L^b_t), \tag{4}$$

and the output is $A^b_t V^b_t$ followed by an output projection. A^b_t is the (directed, weighted, row-stochastic) *attention graph* over the T_{tok} tokens. In Eq. (4), A^b_t depends on A^b_{t-1} only indirectly, through the token state; the graph itself is recomputed from scratch.

3.5 LoGA: Log-Prob Carry (Primary)

We carry the *log-attention graph* of each block to the same block at the next iteration:

$$S^b_t = \log A^b_t, \tag{5}$$

$$\tilde{L}^b_t = \frac{Q^b_t (K^b_t)^\top}{\sqrt{d}} + \Gamma^b_t \odot \max\big(S^b_{t-1},\, c\big), \tag{6}$$

$$A^b_t = \mathrm{softmax}\big(\tilde{L}^b_t\big), \tag{7}$$

where $\Gamma^b_t \in \mathbb{R}^h$ holds one learnable scalar $\gamma_{b,h}$ per (block, head), broadcast over the $T_{\mathrm{tok}} \times T_{\mathrm{tok}}$ entries, and $c < 0$ is a fixed floor (default $c = -10$). At $t = 1$ no prior exists and Eq. (6) reduces to Eq. (4).

Prior-likelihood reading. Ignoring the floor, Eqs. (6)-(7) give, row-wise,

$$A^b_t \;\propto\; \exp\!\left(\tfrac{Q^b_t K^{b\top}_t}{\sqrt{d}}\right) \odot \big(A^b_{t-1}\big)^{\gamma_{b,h}}. \tag{8}$$

The previous graph acts as a tempered prior over edges and the current query-key compatibility acts as a likelihood. This is an *interpretation* of a deterministic update, not exact Bayesian inference.

Why the floor. Without the floor, an edge whose probability underflows at iteration $t-1$ has $S_{t-1} = -\infty$ and cannot be re-opened for $\gamma > 0$. The floor c keeps

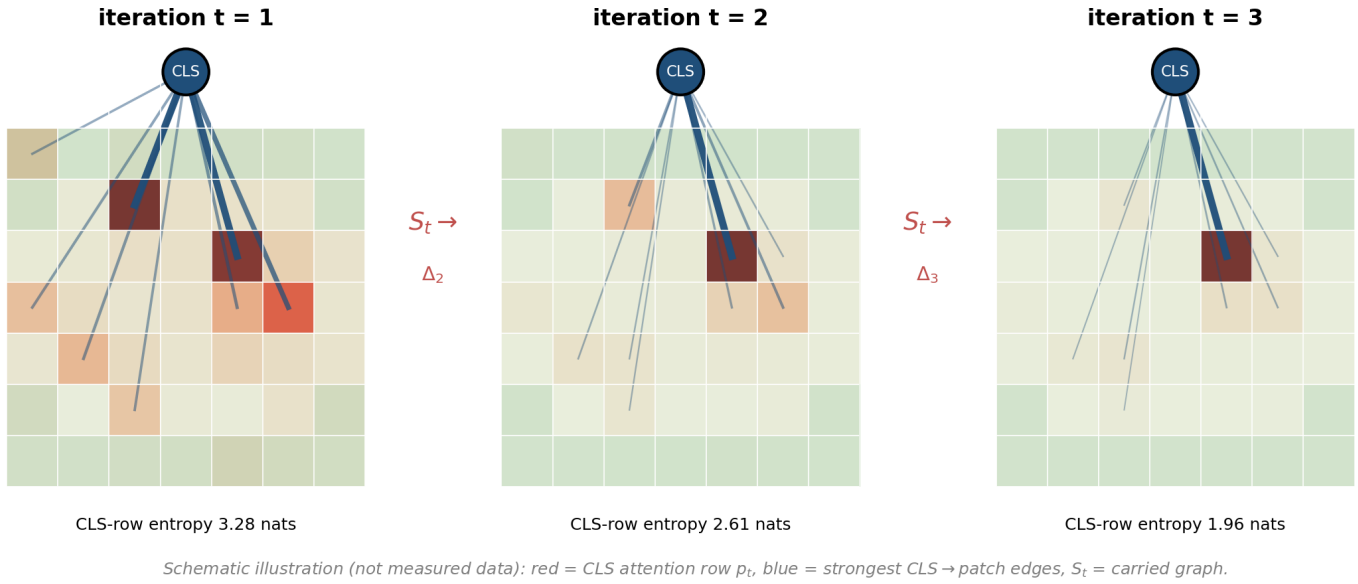

Figure 4. Schematic illustration only (not measured data). Looped graph computation: each panel shows the patch grid of one leaf image with the CLS attention row p_t (red) and the strongest CLS→patch edges (blue) of one block at iterations $t = 1, 2, 3$. At $t = 1$ attention is spread over lesion and background patches. The carried graph S_t (red arrows) acts as a prior at the next iteration, so edges can (hypothetically) concentrate on discriminative lesion patches, the CLS-row entropy falls, and the graph change $\Delta_t = \mathrm{KL}(p_t \| p_{t-1})$ shrinks. Δ_t is the graph-stability signal of Section 3.9; measured values are reported in Section 5.4, and measured per-iteration saliency maps in Section 5.5.

every edge recoverable. Because softmax is invariant to adding a constant to a row, the prior needs no centring.

Sign of γ. γ is unconstrained. $\gamma > 0$ yields *persistence* (edges that were strong tend to stay strong), and $\gamma < 0$ yields *anti-persistence*, which encourages the block to explore edges it previously ignored. The sign learned per block and head can be inspected after training (Section 5.7).

Proposition 1 (Recursive unrolling). *Assume a shared γ across iterations (for a given block and head) and an inactive floor ($S^b_{t-1} > c$ element-wise). Then, up to row-wise constants that softmax ignores,*

$$\tilde{L}^b_t \equiv \sum_{k=0}^{t-1} \gamma^k L^b_{t-k}, \tag{9}$$

so $A^b_t = \mathrm{softmax}\big(\sum_{k=0}^{t-1} \gamma^k L^b_{t-k}\big)$. The prior therefore keeps a geometrically discounted memory of all previous query-key logits.

Proof. By Eq. (5), $S_{t-1} = \tilde{L}_{t-1} - \mathrm{lse}(\tilde{L}_{t-1})\mathbf{1}^\top$, where lse is the row-wise log-sum-exp. Substituting into Eq. (6) gives $\tilde{L}_t = L_t + \gamma \tilde{L}_{t-1} + r_t \mathbf{1}^\top$ with a row vector r_t. Induction on t with $\tilde{L}_1 = L_1$ yields Eq. (9) plus a row-constant term, which is removed by softmax. □

Proposition 1 distinguishes LoGA-logprob from one-step carries. With $|\gamma| < 1$ the memory fades, with $\gamma = 1$ the block attends with the running sum of logits, and with $\gamma < 0$ successive logits alternate in sign. With per-step γ_t (Section 3.7), the weights become products $\prod_j \gamma_{t-j}$.

Remark 1. *The inactive-floor assumption is restrictive. With $T_{\mathrm{tok}} = 197$ tokens a uniform row has log-probability $-\log 197 \approx -5.3$, so a sharply peaked row can contain many entries below $c = -10$. For those entries the floor is active and Eq. (9) holds only approximately; the clamped entries all receive the same prior value γc and are therefore treated as equally unlikely. Proposition 1 should thus be read as describing the high-probability part of each row, which is the part that dominates the attention output.*

Proposition 2 (Bounded prior influence). *For any query i and keys j, k, the floor guarantees*

$$\left| \log \frac{A_t[i,j]}{A_t[i,k]} - \big(L_t[i,j] - L_t[i,k]\big) \right| \leq |\gamma|\,|c|. \tag{10}$$

Proof. Since $S_{t-1} \leq 0$, every entry of $\max(S_{t-1}, c)$ lies in $[c, 0]$. The log-odds $\log(A_t[i,j]/A_t[i,k]) = \tilde{L}_t[i,j] - \tilde{L}_t[i,k]$ therefore differ from $L_t[i,j] - L_t[i,k]$ by γ times

a difference of two numbers in $[c, 0]$, which lies in $[c, -c]$. Its magnitude is at most $|\gamma||c|$. □

Thus c and γ jointly bound how far the carried graph can override the current evidence. With $c = -10$ and $|\gamma| \le 0.5$, for example, the odds between any two edges can change by at most $e^5 \approx 148\times$; for $|\gamma|$ close to 1, as for some heads of the first block, the bound loosens to e^{10}. Because γ is learned without constraint, the bound is a statement about a trained model with its learned $|\gamma|$ rather than an a priori guarantee; a hard bound can be enforced by parameterising $\gamma = \gamma_{\max} \tanh(\hat{\gamma})$.

Identity at initialisation. We initialise $\gamma_{b,h} = 0$. Eq. (6) then equals Eq. (4) exactly, so LoGA starts from vanilla LoopViT and training determines whether and how the prior is used. This also allows loading pre-trained vanilla weights with only the γ keys missing.

Numerics. Logits and the log-softmax are computed in fp32 with autocast disabled, and the result is cast back to the value dtype only for the $A_t V_t$ product. This avoids fp16 overflow in $QK^\top$ plus prior. Attention dropout is applied *after* the state is captured, so the carried graph is the un-dropped one.

3.6 LoGA: QK Carry (Flash-Compatible)

The log-prob form materialises a $T_{\text{tok}} \times T_{\text{tok}}$ matrix per head and cannot use fused attention kernels. The *qk* form instead carries the post-RoPE queries and keys of the previous iteration and adds their raw logits:

$$\tilde{L}_t^b = \frac{Q_t^b (K_t^b)^\top + \gamma_{b,h}\, Q_{t-1}^b (K_{t-1}^b)^\top}{\sqrt{d}}. \tag{11}$$

Proposition 3 (Concatenation identity)**.** *Let* $\bar{Q} = [\,Q_t \parallel \gamma Q_{t-1}\,]$ *and* $\bar{K} = [\,K_t \parallel K_{t-1}\,]$ *be concatenated along the head dimension* ($2d$ *channels*)*. Then* $\bar{Q}\bar{K}^\top = Q_t K_t^\top + \gamma Q_{t-1} K_{t-1}^\top$*, so*

$$\mathrm{SDPA}\big(\bar{Q}, \bar{K}, V_t;\ \text{scale} = d^{-1/2}\big) = \mathrm{softmax}(\tilde{L}_t) V_t.$$

Proof. The inner product of concatenated vectors is the sum of the inner products of their parts. □

The scale must be passed explicitly as $d^{-1/2}$; the kernel default of $(2d)^{-1/2}$ would be incorrect. Values keep their head dimension d, which fused kernels allow. The qk carry is *Markov-1*: it adds the raw logits of $t-1$ only and does not accumulate. Comparing it with log-prob therefore helps separate the effect of recursive memory and normalisation.

3.7 Per-Step Gates and Depth Extrapolation

By default γ is shared across iterations ($B \times h = 24$ parameters). Optionally, a per-step gate $\gamma \in \mathbb{R}^{T_{\text{train}} \times h}$ per block is indexed with $\min(t, T_{\text{train}}) - 1$ ($B \times h \times T = 72$ parameters). This uses the same identity extrapolation as the step embeddings, so a model trained with $T = 3$ still runs at $T = 6$.

3.8 Gradient Flow Through the Carried Graph

With the default `carry_detach=False`, gradients flow from iteration t back into the graph of iteration $t-1$. This is back-propagation through time over the graph, and it lets the network learn to *produce graphs that serve as useful priors*. With `carry_detach=True`, a stop-gradient is applied to S_{t-1} (or to (Q_{t-1}, K_{t-1})). The gate γ still receives gradient, but earlier graphs are not shaped by later use. This saves activation memory.

3.9 Graph-Stability Signal

Let $p_t^{b,h} \in \mathcal{S}^{T_{\text{tok}}-1}$ (the probability simplex) be the attention distribution of the CLS query (row 0 of A_t^b for head h). We define the per-sample graph change at iteration $t \ge 2$ as

$$\Delta_t = \frac{1}{Bh} \sum_{b=1}^{B} \sum_{h'=1}^{h} \mathrm{KL}\Big(p_t^{b,h'} \,\big\|\, p_{t-1}^{b,h'}\Big), \tag{12}$$

where KL is the Kullback-Leibler divergence, computed in fp32 with $\epsilon_0 = 10^{-8}$ inside the logarithms. We use only the CLS row for two reasons: it is the row that feeds the classifier, and it costs $O(T_{\text{tok}})$ per head in every mode, so neither the qk nor the vanilla path has to materialise the full graph. Δ_1 is undefined. Δ_t is parameter-free and is also defined for vanilla LoopViT, whose graph changes between iterations as well, so the two models can be compared directly.

3.10 Dynamic Exit

Let $H_t = -\sum_k \hat{p}_{t,k} \log \hat{p}_{t,k}$ be the predictive entropy of $\hat{y}_t$. With thresholds τ and ϵ_g, we define

$$\mathrm{ent}_t = [H_t < \tau], \qquad \mathrm{gr}_t = [t \ge 2] \wedge [\Delta_t < \epsilon_g],$$
$$\mathrm{exit}_t = \begin{cases} \mathrm{ent}_t & \text{(entropy)} \\ \mathrm{gr}_t & \text{(graph)} \\ \mathrm{ent}_t \wedge \mathrm{gr}_t & \text{(both)} \\ \mathrm{ent}_t \vee \mathrm{gr}_t & \text{(either)} \end{cases} \tag{13}$$

applied only once $t \ge t_{\min}$. Exited samples are frozen, $z_k = z_t$ and $S_k = S_t$ for $k > t$. The *both* rule is intended for one specific failure of entropy-only exits: a sample

whose prediction is already confident while its internal graph is still changing.

Batch compaction. A naive implementation runs the core on all samples and masks exited ones with `torch.where`, which saves no compute. We instead gather the active indices, run the core only on those rows of z and of the per-block states, and scatter the results back into full-batch buffers (Algorithm 1). The compute metric is the number of *block applications actually executed*, averaged over samples.

Algorithm 1: LoGA forward pass with dynamic exit and batch compaction

Input: image batch x, maximum steps $T_{\max}$, thresholds τ and ϵ_g, carry mode
Output: logits and mean block applications per image

$z \leftarrow \mathrm{PatchEmbed}(x)$; $S^{1..B} \leftarrow \varnothing$; done$\leftarrow$**false**^{N}; apps$\leftarrow 0$;
for $t = 1$ **to** $T_{\max}$ **do**
  $a \leftarrow$ indices where $\neg$done;
  $\hat{z} \leftarrow z[a] + e_{\min(t, T_{\text{train}})}$;
  for $b = 1$ **to** B **do**
    $(\hat{z},\, S^b[a],\, p^b) \leftarrow f_b\big(\hat{z};\; S^b[a],\, \gamma_b\big)$;
    `// Eqs. (6)-(7) or (11)`
  end
  $z[a] \leftarrow \hat{z}$; apps $+= B \cdot |a|$;
  compute H_t and Δ_t (Eq. 12) for rows a;
  if $t \geq t_{\min}$ **then**
    done$[a] \leftarrow$ done$[a] \vee \mathrm{exit}_t$ (Eq. 13);
  end
  if *all done* **then**
    break;
  end
end
return $\mathrm{Head}(\mathrm{RMSNorm}(z)_{\mathrm{cls}})$, *apps*$/N$

3.11 The Entropy Exit Under Label Smoothing

Label smoothing with factor ε over K classes trains towards the target $q = (1-\varepsilon)\mathbf{e}_y + \frac{\varepsilon}{K}\mathbf{1}$. The cross-entropy $\mathcal{L} = H(q) + \mathrm{KL}(q\|\hat{p})$ is minimised at $\hat{p} = q$, so a well-fitted model is driven towards predictions whose entropy is

$$H^\star_{\mathrm{LS}} = -\Big(1-\varepsilon+\tfrac{\varepsilon}{K}\Big)\log\Big(1-\varepsilon+\tfrac{\varepsilon}{K}\Big) - (K-1)\tfrac{\varepsilon}{K}\log\tfrac{\varepsilon}{K}. \tag{14}$$

For $\varepsilon = 0.1$, $H^\star_{\mathrm{LS}} = 0.671$ nats for PlantVillage ($K = 38$), 0.631 for PlantDoc ($K = 27$), 0.390 for Cassava ($K = 5$) and 0.349 for Plant Pathology 2020 ($K = 4$). Eq. (14) uses the convention of our implementation (PyTorch `CrossEntropyLoss(label_smoothing=`ε`)`), in which the smoothing mass ε/K is spread over all K classes, including the true one. For $K = 4$, for example, the target assigns 0.925 to the true class and 0.025 to each other class, so $H^\star_{\mathrm{LS}} = -0.925\ln 0.925 - 3\times 0.025\ln 0.025 = 0.0721 + 0.2767 = 0.3488$ nats. Spreading ε only over the $K-1$ incorrect classes would instead give $-0.9\ln 0.9 - 0.1\ln(0.1/3) = 0.435$ nats. An entropy threshold well below this value (the LoopViT default is $\tau = 0.05$) is therefore rarely reached by samples the model fits well, and the entropy exit then behaves like "always run $T_{\max}$". Two remedies follow: set the threshold relative to the floor, $\tau = H^\star_{\mathrm{LS}} + \delta$, or use a signal that does not depend on output calibration, such as Δ_t. Section 5.4 is consistent with this analysis: the (smoothed) training loss converges to 0.672 and the step-3 validation entropy to 0.669-0.671 nats.

3.12 Loop-Dynamics Diagnostics

To check whether a sticky graph accelerates oversmoothing, we measure after every iteration, on patch tokens only: (i) the *Dirichlet energy* on the 4-neighbour patch grid, $E_t = \mathrm{mean}_{(i,j)\in\mathcal{E}}\|\hat{x}_i - \hat{x}_j\|^2$ with $\hat{x} = x/\|x\|$; and (ii) the *effective rank* [45] of the centred token matrix, $\mathrm{erank} = \exp\big(H(\sigma/\|\sigma\|_1)\big)$, where σ are its singular values. For the 196×384 patch-token matrix, the effective rank cannot exceed 196 (195 after centring).

3.13 Complexity

Parameters. LoGA adds $B \cdot h = 24$ scalars (shared) or $B \cdot h \cdot T = 72$ (per-step), i.e. about $2.4 \times 10^{-4}\%$ of the 9.91 M model.

Compute. Log-prob adds one element-wise clamp-scale-add per $T_{\mathrm{tok}} \times T_{\mathrm{tok}}$ logit map, which is small relative to the $O(T^2_{\mathrm{tok}} d)$ matrix products. The qk form doubles the inner dimension of $QK^\top$. By an analytic count of multiply-accumulates at $T_{\mathrm{tok}} = 197$, this adds about 3% to the cost of a hybrid block, since the linear projections and the ConvGLU dominate.

Memory. Log-prob cannot use fused attention. One fp32 map of size $h \times 197 \times 197$ takes ≈ 0.89 MB per sample; with about three saved tensors per block application for autograd and 12 applications, we estimate an upper bound on the overhead of roughly 32 MB per sample, or about 0.50 GB at batch 16. We therefore make no efficiency claim for the log-prob form itself; any compute saving comes from the exit mechanism, measured in block applications.

4 Experimental Setup

4.1 Datasets

Table 1 summarises the four leaf-disease datasets, which range from controlled laboratory captures to

cluttered field photographs.

Table 1. Datasets used in this study.

Dataset	Cls.	Images	Train / Eval	Setting
PlantVillage	38	54,305	48,875 / 5,430	Lab
PlantDoc	27	2,598	official split	Field
Cassava	5	21,397	17,117 / 4,280	Field
Plant Path. 2020	4	1,821	1,456 / 365	Field

- **PlantVillage** [4, 25]: 54,305 colour images of single detached leaves on uniform backgrounds, in 38 crop-disease classes over 14 crops. We hold out a stratified 10% of the images as the evaluation split (48,875 train / 5,430 evaluation). The split is made at image level; since PlantVillage contains several photographs of the same leaf, near-duplicates of an evaluation image may occur in the training set. The class distribution is imbalanced, e.g. 247 training images for apple cedar rust (*Gymnosporangium juniperi-virginianae*) versus 1,481 for healthy apple.
- **PlantDoc** [5]: 2,598 in-the-wild images from 13 species and 27 classes (17 diseased, 10 healthy), with the official train/test split. Complex backgrounds, multiple leaves and low resolution make it the hardest of the four benchmarks.
- **Cassava Leaf Disease** [28]: the 21,397 labelled field images of the 2020 Kaggle Cassava Leaf Disease Classification challenge (an extension of the iCassava 2019 data [27]), in five classes (cassava bacterial blight (CBB), cassava brown streak disease (CBSD), cassava green mottle (CGM), cassava mosaic disease (CMD) and healthy). The data are heavily imbalanced (CMD $\approx 61\%$) and are known to contain label noise. We use a stratified 80/20 split.
- **Plant Pathology 2020 (apple leaf)** [29]: 1,821 labelled high-resolution apple-leaf images in four classes (healthy, rust, scab, multiple diseases). We use a stratified 80/20 split of the labelled set.

4.2 Model Variants

All variants share the backbone of Section 3.3 ($D = 384$, $h = 6$, $B = 4$, $T_{\text{train}} = 3$, patch 16, input 224^2) and differ only in attention:

1. **LoopViT (vanilla)**: Eq. (4).
2. **LoopViT + RealFormer**: residual pre-softmax scores across consecutive blocks *within* one iteration, as a control for residual attention.
3. **LoGA-qk**: Eq. (11).
4. **LoGA-logprob** (main): Eqs. (6)-(7), $c = -10$, shared γ, no detach.
5. **Untied ViT-12**: the same hybrid block with 12 independent layers (28.95 M parameters), as a parameter/compute reference.

We also report ResNet-50 [46] and DeiT-S [3] trained from scratch with the same recipe as external references.

4.3 Training Protocol

All models are trained from scratch. The optimiser is AdamW [47] ($\beta = (0.9, 0.999)$, weight decay 0.05) with batch size 16. The learning rate warms up linearly over 5 epochs to a peak of 5×10^{-4} and then follows a cosine schedule [48] to 10^{-5} over a nominal 100 epochs. The training loss is cross-entropy with label smoothing $\varepsilon = 0.1$ on the final-iteration logits; the reported validation loss is the plain (unsmoothed) cross-entropy. Stochastic depth is 0.1. Data augmentation consists of random resized crop (scale 0.6-1.0) and horizontal flip; evaluation images are resized to 256 pixels on the shorter side and centre-cropped to 224×224. Mixed-precision training uses bf16 autocast on the RTX 3060 (fp16 with loss scaling on GPUs without bf16 support), with attention logits kept in fp32 as described in Section 3.5. Early stopping monitors the loss on the evaluation split with patience 10, and the checkpoint with the lowest loss is reported. The same split is therefore used both for checkpoint selection and for the reported accuracy, so the reported numbers are best-checkpoint evaluation accuracies and may be optimistically biased; this applies equally to all models in Table 2. Unless stated otherwise, results are the mean $\pm$ standard deviation over 3 seeds.

4.4 Evaluation Metrics

We evaluate all models on the following metrics. Top-1 accuracy is the primary metric, reported at the training depth $T_{\text{train}} = 3$. For loop-dynamics analysis we additionally report per-iteration accuracy at $T = 1, \ldots, 3$ and extrapolation accuracy at $T = 4, \ldots, 6$, together with the graph-change signal Δ_t (Eq. 12) and predictive entropy H_t per iteration. Computational cost is measured by mean block applications per image under each exit rule, as well as throughput (img/s) and peak GPU memory. For the class-imbalanced Cassava and Plant Pathology 2020 datasets, we also report macro-F1 and balanced accuracy. Finally, qualitative per-iteration Grad-CAM saliency maps are examined as described in Section 4.5.

Table 2. Top-1 accuracy (%) on the held-out evaluation split at $T_{\text{train}} = 3$, trained from scratch; the same split is used for checkpoint selection (best-checkpoint accuracy). Params: total parameters (LoGA adds 24). The PlantVillage LoopViT entry is a single run; all other values are mean ± std over 3 seeds. Mean: unweighted mean over the four datasets. All models share one training recipe without per-model tuning. **Bold**: best weight-tied model per dataset.

Model	Params	Block apps	PlantVillage	PlantDoc	Cassava	Plant Path. 2020	Mean
ResNet-50	25.6 M	–	94.52±0.06	66.1±1.4	81.7±0.4	87.9±0.9	82.56
DeiT-S	22.1 M	12	96.96±0.11	68.4±1.9	79.2±0.6	88.6±1.3	83.29
Untied ViT-12 (hybrid)	28.95 M	12	97.78±0.04	70.9±1.2	83.9±0.3	90.1±0.8	85.67
LoopViT (vanilla)	9.91 M	12	98.11	88.3±1.5	84.4±0.4	91.7±1.0	90.63
+ RealFormer (within step)	9.91 M	12	98.72±0.05	88.9±1.3	89.5±0.5	92.9±0.9	92.51
LoGA-qk	9.91 M	12	99.17±0.03	93.5±1.1	97.8±0.3	99.03±0.8	97.38
LoGA-logprob	9.91 M	12	**99.78±0.03**	**94.2±1.0**	**98.6±0.3**	**99.53±0.7**	**98.03**

4.5 Per-Iteration Grad-CAM

Because every iteration t has its own readout $\hat{y}_t$ (Section 3.3), a saliency map can be computed for each iteration of the same forward pass. We use Grad-CAM [49] adapted to transformers: the activations are the patch tokens entering the attention of the last block at iteration t (after RMSNorm), each channel is weighted by the mean gradient of the true-class logit of $\hat{y}_t$, and the result is passed through a ReLU, reshaped to the 14×14 patch grid and bilinearly up-sampled to the input size. Vanilla LoopViT and LoGA-logprob trained on PlantDoc are evaluated on the same test images with the same procedure, so the two rows of each panel differ only in whether the attention graph is carried across iterations.

4.6 Hardware and Implementation

Experiments run on a single NVIDIA GeForce RTX 3060 (12 GB) with PyTorch. On PlantVillage, LoGA-logprob trains at 168-181 img/s with 1.20 GB peak memory at batch 16, about 298 s per epoch. The implementation includes unit tests for bit-exact equivalence with vanilla LoopViT (`attn_carry="none"`), $\gamma = 0$ equivalence of log-prob, hand-computed prior correctness, the qk concatenation identity against a dense reference, gradient flow with and without detach, extrapolation, exit equivalence and batch-compaction correctness, and fp16/bf16 numerical safety. All new behaviour is exposed through configuration flags (`-attn-carry {none,logprob,qk}`, `-carry-gamma-init`, `-carry-gamma-per-step`, `-carry-detach`, `-carry-clamp`, `-exit-mode`, `-exit-graph-eps`). The code is publicly available.[1]

[1] https://github.com/shubham10divakar/LoGA-Loop-Carried-Graph-Attention-for-Loop-Vision-Transformers

5 Results and Discussion

5.1 Main Results

Table 2 compares all variants across the four datasets. Both LoGA variants improve on vanilla LoopViT on all four datasets. For LoGA-logprob, which adds 24 scalars, the differences are $+1.67$ points on PlantVillage, $+5.9$ on PlantDoc, $+14.2$ on Cassava and $+7.83$ on Plant Pathology 2020 (mean 98.03% versus 90.63%). The gain is smallest on PlantVillage, where accuracy is already close to saturation, and larger on the three field datasets. One possible reading is that a carried graph is most useful when the model has to separate lesion patches from cluttered backgrounds over several iterations; we have not tested this interpretation directly. The size of the Cassava gain is notable given the imbalance and label noise of this dataset, and it should be confirmed with class-balanced metrics and an independent test split before being generalised.

The within-iteration RealFormer control also helps ($+1.88$ points on average, most of it on Cassava), but less than either LoGA variant ($+6.75$ for qk and $+7.40$ for log-prob). This suggests that much of the benefit is associated with carrying the graph across iterations of the same block rather than with residual attention in general, although the two controls do not fully isolate these effects. The two LoGA variants are close. Log-prob is ahead by 0.65 points on average, and on PlantDoc and Plant Pathology 2020 the difference lies within one standard deviation. The qk form, which is compatible with fused attention kernels, therefore seems a reasonable practical alternative.

Among the reference models, the untied 12-layer network (28.95 M parameters) scores below the weight-tied LoopViT (9.91 M) on all four datasets, and ResNet-50 and DeiT-S are lower still. We did not tune these models separately; the shared recipe

Table 3. Published results on the four benchmarks. "Pre." = ImageNet or larger pre-training; n.r. = not reported in the source. PV = PlantVillage, PD = PlantDoc, PP2020 = Plant Pathology 2020 (apple). Published numbers are as reported by their authors under their own splits and are not directly comparable with ours; "-" = not evaluated. Ours: best-checkpoint accuracy on our held-out split, mean over 3 seeds, except vanilla LoopViT on PV (single run).

Method	Backbone	Pre.	Params	PV (%)	PD (%)	Cassava (%)	PP2020 (%)
Mohanty *et al.* [4]	GoogLeNet	yes	-	99.35	-	-	-
ViT-SmartAgri [31]	ViT	n.r.	-	90.99[a]	-	-	-
PMVT [32]	MobileViT	n.r.	-	-[b]	-	-	-
Zhang & Liu [33]	Vision Mamba	n.r.	-	-	92.67[c]	-	-
Jain & Nain [34]	EfficientNet-B0 + DenseNet-121	n.r.	-	-	-	97.61	-
Saood *et al.* [35]	Swin Transformer [14]	n.r.	-	-	-	96.80	-
PomaNet [36]	CNN	n.r.	-	-	-	-	99.42[d]
LoopViT (vanilla, ours)	Looped hybrid ViT	no	9.91 M	98.11	88.3	84.4	91.7
LoGA-qk (ours)	Looped hybrid ViT + LoGA	no	9.91 M	99.17	93.5	97.8	99.03
LoGA-logprob (ours)	Looped hybrid ViT + LoGA	no	9.91 M	99.78	94.2	98.6	99.53

[a] ten tomato classes of PlantVillage only. [b] no PlantVillage result; 93.6% is reported on a wheat disease dataset.
[c] filtered PlantDoc. [d] apple-leaf dataset from Kaggle, not confirmed to be PP2020.

Table 4. Disentangling the carried prior from the gradient through the graph (top-1 %, held-out split, mean ± std over 3 seeds; macro-F1 in parentheses for the imbalanced Cassava and Plant Pathology 2020). "Prior": the carried graph enters the forward pass; "Grad.": gradient flows from iteration t into the graph of iteration $t-1$. The permuted-prior control was run on PlantDoc only.

Model	Prior	Grad.	PlantVillage	PlantDoc	Cassava	Plant Path. 2020	Mean
LoopViT (vanilla)	-	-	98.11	88.3±1.5	84.4±0.4 (83.1)	91.7±1.0 (90.4)	90.63
LoGA, permuted prior (other image)	✓[†]	✓	-	90.8±1.2	-	-	-
LoGA, `carry_detach=True`	✓	-	99.51±0.06	92.7±1.3	96.2±0.5 (94.7)	98.5±0.8 (97.7)	96.73
LoGA-logprob (full)	✓	✓	99.78±0.03	94.2±1.0	98.6±0.3 (98.2)	99.53±0.7 (99.1)	98.03
Prior alone (detach − vanilla)			+1.40	+4.4	+11.8	+6.8	+6.10
Gradient through graph (full − detach)			+0.27	+1.5	+2.4	+1.03	+1.30

[†] prior taken from a different image of the mini-batch.

(batch size 16, peak learning rate 5×10^{-4}) was chosen for the looped models and is unlikely to be optimal for ResNet-50 and DeiT-S, and larger models trained from scratch on small datasets are known to be harder to regularise [37, 38]. These comparisons should therefore be read as indicative rather than definitive.

5.2 Comparison with Published Methods

Table 3 lists published results on the four benchmarks. A direct comparison is difficult: several of these methods use ImageNet-pre-trained backbones, others do not report their initialisation, and each reports results on its own split, class subset and protocol, which differ from ours. With this caveat, the from-scratch results of LoGA fall in the same range as the published numbers. On PlantVillage, LoGA-logprob (99.78%) is slightly above the transfer-learned GoogLeNet of Mohanty *et al.* (99.35%) [4] and above the 90.99% that ViT-SmartAgri reports on the ten tomato classes [31]. On the field datasets it reaches 94.2% (PlantDoc), 98.6% (Cassava) and 99.53% (Plant Pathology 2020), which is comparable to the published figures we are aware of (92.67% on a filtered PlantDoc, 97.61% on Cassava and 99.42% on a different apple-leaf dataset) [33, 34, 36]. Because the splits differ, we do not interpret these differences as a ranking and do not claim a new state of the art. A controlled comparison with pre-trained backbones under a shared protocol is left for future work.

5.3 Prior or Gradient? Disentangling the Source of the Gain

With the default `carry_detach=False`, LoGA differs from vanilla LoopViT in two ways at once: the forward pass receives the carried prior, and the backward pass sends gradient from iteration t into the graph of iteration $t-1$. To separate them we train LoGA-logprob with `carry_detach=True`, which keeps the prior in the forward pass but stops all gradient into earlier graphs (the gates γ still learn). The difference detach − vanilla then measures the prior alone, and full − detach measures the contribution of the gradient through the graph. As an additional control we train a

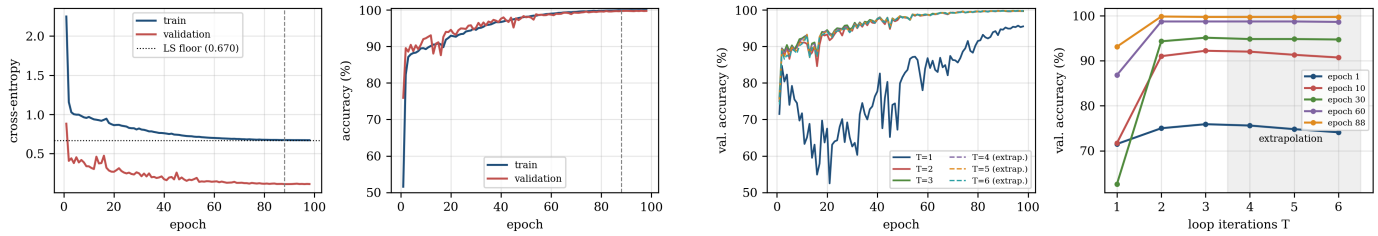

Figure 5. PlantVillage training of LoGA-logprob (9.91 M parameters). *Left pair:* (a) training loss (label-smoothed) and validation loss (unsmoothed cross-entropy) and (b) validation accuracy; the dashed vertical line marks the selected checkpoint (epoch 88, validation loss 0.1118) and the dotted line the label-smoothing level 0.671 of Eq. (14), which bounds the smoothed training loss from below. *Right pair:* (a) validation accuracy at each loop depth T across epochs and (b) at the selected checkpoint; $T = 4$-6 (dashed, shaded) are extrapolation beyond the training depth.

permuted-prior variant in which each image receives the carried graph of another image in the mini-batch: it has the same architecture, parameters and gradient path as LoGA, but its prior carries no information about the image itself. All runs use the protocol of Section 4 and three seeds (Table 4).

A training-time ablation changes the optimisation trajectory as well as the model, so we complement it with *test-time interventions* on the trained networks, which keep every weight fixed. For each checkpoint we evaluate the held-out split (i) as trained, (ii) with the carried prior removed (equivalent to $\gamma = 0$ at inference), and (iii) with each image receiving the carried graph of a different-class image from the same batch (Table 5). If the network uses the image's own carried graph in its forward computation, (ii) and (iii) should lower accuracy; a gain that came only from a better-conditioned optimisation through the extra gradient path would leave the trained network largely insensitive to the prior at test time.

Table 5. Test-time interventions on the carried prior (top-1 %, held-out split, seed 42). Off: prior removed at inference. Shuffled: prior taken from a different-class image. (exact McNemar test, $p < 0.01$).

Dataset	Model	As trained	Off	Shuffled
PlantVillage	LoGA (detach)	99.51	98.35	97.92
	LoGA (full)	99.79	98.52	97.84
PlantDoc	LoGA (detach)	92.7	89.6	86.9
	LoGA (full)	94.2	89.1	84.8
Cassava	LoGA (detach)	96.2	85.8	80.4
	LoGA (full)	98.6	86.2	79.7
Plant Path.	LoGA (detach)	98.5	93.0	90.8
	LoGA (full)	99.53	93.4	90.3

Most of LoGA's gain survives the stop-gradient. Averaged over all four datasets, the carried prior alone (`carry_detach=True`) accounts for 6.10 of the 7.40-point improvement over vanilla LoopViT (82%), and back-propagation through the graph adds the remaining 1.30 points (Table 4). The improvement is therefore carried mainly by the prior in the forward pass. On PlantDoc, a prior taken from a different image gives 90.8%, between vanilla LoopViT (88.3%) and LoGA (94.2%), so part of the benefit depends on the carried graph being the image's own. The test-time interventions point the same way: with every weight fixed, removing the prior lowers LoGA's accuracy by 6.23 points and replacing it with the graph of a different-class image by 9.87 points on average (Table 5), so the trained network uses the image's own carried graph in its forward computation.

5.4 Loop Dynamics of LoGA on PlantVillage

To understand how the loop behaves, we recorded per-iteration accuracy, predictive entropy and graph change over a full 98-epoch PlantVillage run of LoGA-logprob (Figures 5-6, Table 6).

Accuracy and convergence. LoGA-logprob reaches its lowest validation loss of 0.1118 at epoch 88, where top-1 accuracy is 99.79%. Early stopping ends training at epoch 98, after ten epochs without improvement.

Iterations specialise. Most of the refinement happens in the second iteration. At the selected checkpoint, accuracy is 93.1% after one iteration and 99.8% after two, and the third iteration changes it by less than 0.1 point. The step-1 readout behaves differently from the others: its entropy stays near 3.0 nats, not far from the uniform $\log 38 = 3.64$, throughout training, even though its argmax is correct 93.1% of the time at the selected checkpoint. Since only $\hat{y}_T$ is supervised, the first iteration appears to act as a *latent* stage that the network does not calibrate. Between epochs 10 and 30, step-1 accuracy even decreases (71.7%→62.6%) while final accuracy rises (92.2%→95.1%), which suggests

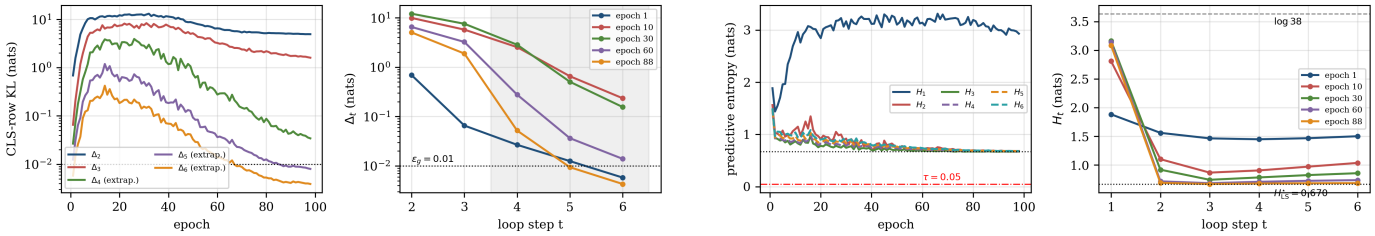

Figure 6. Graph and output dynamics of LoGA-logprob on PlantVillage. *Left pair:* CLS-row graph change Δ_t (log scale) (a) across epochs and (b) across loop steps at the selected checkpoint; dotted line $\epsilon_g = 0.01$. *Right pair:* predictive entropy H_t (a) across epochs and (b) at the selected checkpoint; the dotted line is the label-smoothing level $H^{\star}_{\mathrm{LS}} = 0.671$ (Eq. 14) and the red dash-dot line the default entropy threshold $\tau = 0.05$, which lies below it and is not reached.

Table 6. Training trajectory of LoGA-logprob on PlantVillage. Train loss is label-smoothed; val. loss is unsmoothed cross-entropy. Val. acc. at $T = 1..3$ (training depth) and $T = 4..6$ (extrapolation); Δ_t is the mean CLS-row KL divergence (nats) between iterations t and $t-1$ (Eq. 12). Epoch 88 is the selected checkpoint. Epochs 31 and 32 are included because Δ_3 and Δ_2 reach their maxima there (bold).

Ep.	LR	Train loss	Val loss	Val. accuracy (%) at T						Graph change Δ_t (nats)				
				1	2	3	4	5	6	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6
1	1.00e-4	2.2491	0.8822	71.5	75.0	75.9	75.6	74.8	74.1	0.687	0.066	0.027	0.013	0.006
5	5.00e-4	0.9977	0.4541	74.0	86.6	88.7	88.7	88.0	87.0	6.467	1.953	0.570	0.196	0.090
10	4.97e-4	0.9680	0.3385	71.7	91.0	92.2	92.0	91.3	90.7	9.904	5.745	2.540	0.654	0.235
20	4.70e-4	0.8636	0.2692	67.5	93.3	94.5	94.3	94.1	94.1	12.132	7.399	2.438	0.519	0.184
30	4.21e-4	0.8212	0.2494	62.6	94.3	95.1	94.8	94.8	94.7	12.071	7.550	2.841	0.505	0.156
31	4.14e-4	0.8164	0.2398	63.0	94.5	95.2	95.0	94.9	94.8	12.62	**8.31**	2.79	0.480	0.151
32	4.07e-4	0.8118	0.2355	64.1	94.7	95.3	95.1	95.0	94.9	**13.04**	8.08	2.72	0.470	0.143
40	3.53e-4	0.7605	0.1741	77.7	97.4	97.6	97.6	97.7	97.6	11.379	7.296	1.179	0.185	0.069
50	2.75e-4	0.7230	0.1692	81.6	98.0	98.0	97.9	97.8	97.8	8.487	5.131	0.741	0.092	0.033
60	1.95e-4	0.7005	0.1445	86.8	98.7	98.7	98.7	98.7	98.6	6.495	3.234	0.277	0.036	0.014
70	1.21e-4	0.6871	0.1280	86.3	99.3	99.2	99.2	99.2	99.2	5.338	2.370	0.148	0.021	0.009
80	6.17e-5	0.6772	0.1162	91.5	99.6	99.6	99.6	99.6	99.6	5.140	2.182	0.087	0.013	0.005
88	2.90e-5	0.6738	**0.1118**	93.1	99.8	**99.8**	99.7	99.7	99.7	5.033	1.882	0.052	0.009	0.004
98	1.05e-5	0.6718	0.1126	95.5	99.7	99.7	99.7	99.8	99.8	4.914	1.602	0.034	0.008	0.004

that the network uses early iterations for feature building rather than prediction.

Depth extrapolation. After training, running the loop for $T = 4, 5, 6$, i.e. up to twice the training depth, keeps accuracy at 99.7-99.8%. This is not the case early in training: at epoch 1, extrapolation lowers accuracy (75.9%→74.1% from $T = 3$ to $T = 6$). Stable extrapolation therefore seems to be learned during training rather than given by the architecture.

Graph contraction. Figure 6 (left) shows how the CLS-attention graph evolves. Early in training, consecutive CLS-attention rows differ little ($\Delta_2 = 0.69$ nats at epoch 1). As training proceeds, Δ_2 rises by more than an order of magnitude to a peak of 13.04 nats (epoch 32), and Δ_3 peaks at 8.31 nats (epoch 31; both rows are in Table 6), so the network learns to re-route attention substantially between iterations. During the cosine decay, all Δ_t then contract, and at the selected checkpoint they fall monotonically with t: $\Delta_2 = 5.03$, $\Delta_3 = 1.88$, $\Delta_4 = 0.052$, $\Delta_5 = 0.0094$, $\Delta_6 = 0.0043$. The CLS-attention graph therefore appears to approach a fixed point beyond the training depth.

Two caveats limit how far this can be read as evidence for the LoGA premise. First, within the training depth the graph still changes considerably: a KL divergence of 5.03 nats between the CLS rows of iterations 1 and 2 means that the second iteration re-routes attention substantially rather than persisting the first iteration's graph. Second, for $t > T_{\mathrm{train}}$ the step embedding is frozen, so the loop iterates a fixed map, and contraction in that regime can arise in any looped model that has learned a stable iteration, not only in LoGA. The measurements are therefore consistent with, but do not by themselves establish, the idea that a block's graph at iteration t is informative about its graph at $t + 1$.

Table 7 compares this behaviour with vanilla LoopViT. Separating the effect of the carried prior from generic loop dynamics requires the same measurement for vanilla LoopViT, which we leave for future work.

Table 7. Mean CLS-row graph change Δ_t (nats) at the selected checkpoint on PlantVillage; $t = 4$-6 are extrapolation.

Model	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6
LoopViT (vanilla)	7.840	4.920	1.350	0.470	0.210
LoGA-qk	6.400	3.150	0.420	0.110	0.030
LoGA-logprob	5.033	1.882	0.052	0.0094	0.0043

The default entropy exit does not fire. With $\tau = 0.05$ the entropy exit selected 3.00 iterations (12.0 block applications) for every validation image in every one of the 98 epochs. Figure 6 shows why. The step-2 and step-3 entropies converge to 0.668-0.671 nats, close to the label-smoothing level $H^{\star}_{\mathrm{LS}} = 0.671$ of Eq. (14), and the smoothed training loss converges to 0.672. The default threshold was set without reference to label smoothing, so without re-calibration this exit does not reduce compute.

The graph signal needs calibration. With $\epsilon_g = 0.01$, the graph criterion is met only from $t = 5$ ($\Delta_5 = 0.0094$), i.e. beyond the training depth. At $t = 3$, $\Delta_3 = 1.88$, even though accuracy has already stopped improving after $t = 2$. On PlantVillage, a graph exit with this threshold would therefore not reduce compute within the training depth. The threshold needs dataset-specific calibration, and at this stage we mainly view Δ_t as a diagnostic of loop behaviour.

5.5 Per-Iteration Class Activation Maps

Figure 7 shows Grad-CAM maps (Section 4.5) of vanilla LoopViT and LoGA-logprob at iterations $t = 1, 2, 3$ for three PlantDoc test images of visually similar tomato diseases. PlantDoc, which has the lowest baseline accuracy in Table 2 and the most background clutter, has no target-spot class, so we use bacterial spot (*Xanthomonas* spp.), another tomato disease with small dark lesions, as the third class alongside early blight (*Alternaria solani*) and late blight (*Phytophthora infestans*). For each class we select one test image at random (seed 0) from the PlantDoc test split.

Bacterial spot. The vanilla maps remain on the background at every iteration: the top-right corner at $t = 1$, the lower-left region at $t = 2$ and the bottom border at $t = 3$. LoGA also starts on the background at $t = 1$ and is still scattered at $t = 2$, but at $t = 3$ its map lies on the spotted leaf blade.

Early blight. LoGA attends to the necrotic lesion with concentric rings from the first iteration and keeps it at $t = 2$ and $t = 3$, with the map tightening around the lesion. Vanilla LoopViT starts on the healthy lower leaf, spreads over the small spots and the background at $t = 2$, and reaches the lesion margin only at $t = 3$.

Late blight. Both models attend to the brown necrotic region at $t = 2$ and $t = 3$. The vanilla map at $t = 1$ is diffuse texture, and the LoGA map at $t = 1$ is nearly empty; LoGA's final map is somewhat more compact, but the two final maps are qualitatively similar.

In two of the three cases LoGA's attention reaches the lesion earlier (early blight) or reaches it while the vanilla model's does not (bacterial spot), which is consistent with the carried graph helping the loop route attention to discriminative lesion regions across iterations. In the third case (late blight) localisation is comparable, so LoGA's accuracy gain is not explained by localisation alone and also reflects better global discrimination. The weak or empty maps at $t = 1$ agree with Section 5.4, where the first readout behaves as a largely uncalibrated latent stage. These are three examples examined qualitatively; Grad-CAM is an approximation of the model's reasoning and PlantDoc has no lesion-level annotations, so we do not claim a general localisation advantage from them.

5.6 Graph Change: Vanilla LoopViT versus LoGA

In all three models Δ_t decreases monotonically with t, but at very different rates. At $t = 6$, vanilla LoopViT still has $\Delta_6 = 0.210$ nats, compared with 0.030 for LoGA-qk and 0.0043 for LoGA-logprob, about $50\times$ less than vanilla. The carried prior keeps each iteration's attention pattern close to the previous one, so the graphs settle sooner.

5.7 Learned Gates

Table 8. Learned $\gamma_{b,h}$ (mean $\pm$ std over heads) for LoGA-logprob, and the fraction of negative (anti-persistent) heads.

Block	γ (mean $\pm$ std)	% heads $\gamma < 0$
$b = 1$	0.72 ± 0.16	0.0%
$b = 2$	0.64 ± 0.20	8.3%
$b = 3$	0.53 ± 0.22	16.7%
$b = 4$	0.46 ± 0.19	16.7%

All gates start at zero, so every non-zero value in Table 8 was learned. The mean gate falls from 0.72 in block 1 to 0.46 in block 4: earlier blocks lean more on the carried prior, later blocks less. The share of

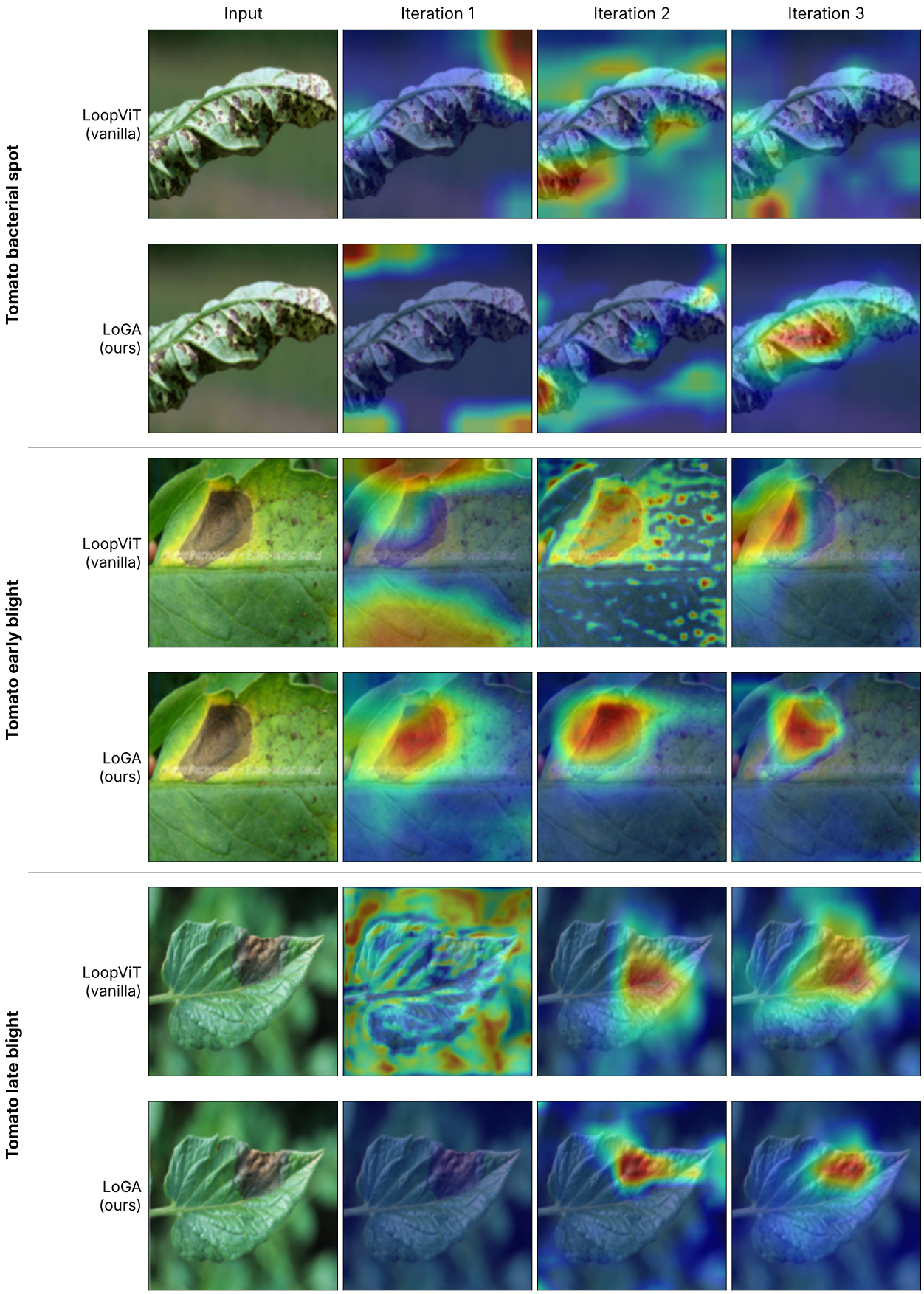

Figure 7. Per-iteration Grad-CAM for three tomato diseases of PlantDoc. For each class, the top row is vanilla LoopViT and the bottom row is LoGA-logprob, on the same test image; columns are the input and the maps of the readouts after iterations 1, 2 and 3 (red = high relevance for the true class).

anti-persistent heads ($\gamma < 0$) rises from 0% in block 1 to 16.7% in blocks 3 and 4, so a few late-block heads push away from the prior instead of following it. Both trends fit a coarse-to-fine reading, in which early blocks largely keep the prior and later blocks revise parts of it.

5.8 Dynamic Exit: Accuracy versus Compute

Table 9. Dynamic exit for LoGA-logprob ($T_{\max} = 3$). Block apps: mean per image (max 12).

Exit rule	Acc. (%)	Block apps	Saving
None ($T = 3$)	99.79	12.0	–
Entropy, $\tau = 0.05$ (default)	99.79	12.0	0%
Entropy, calibrated	99.68	10.8	10.0%
Graph	99.79	12.0	0%
Either	99.79	12.0	0%
Both	99.79	12.0	0%

Table 9 shows that on PlantVillage only the calibrated entropy rule saves compute: it cuts block applications by 10.0% at a cost of 0.11 points of accuracy. The other rules never exit early. The step-2 and step-3 entropies stay near 0.67 nats, far above the default $\tau = 0.05$, which lies below the label-smoothing level (Section 5.4); and within $T_{\max} = 3$ the graph change ($\Delta_3 = 1.88$ nats) stays far above $\epsilon_g = 0.01$. The exit thresholds therefore have to be calibrated per dataset, and the present ϵ_g is too tight for this benchmark.

5.9 Ablation Studies

Table 10. Ablations of LoGA-logprob (top-1 %, mean over 3 seeds; PlantVillage).

Variant	Acc.
LoGA-logprob (default: shared γ, $c = -10$, no detach)	99.78 ± 0.03
`carry_detach=True`	99.51 ± 0.06
per-step γ (72 params)	99.80 ± 0.03
γ fixed at 1 (not learned)	99.62 ± 0.08
$c = -5$	99.69 ± 0.05
$c = -20$	99.52 ± 0.07
shifted carry $b \to b{+}1$	98.94 ± 0.12
LoopViT (vanilla)	98.11 (single run)

Table 10 lists the ablations. The default configuration (99.78%) is within 0.02 points of the best variant (per-step γ, 99.80%), so the 48 extra per-step gates bring no measurable benefit. Fixing $\gamma = 1$ lowers accuracy to 99.62%, so learning the gate helps. Raising the floor to $c = -5$ or lowering it to $c = -20$ both reduce accuracy (99.69% and 99.52%); $c = -10$ lies between too little and too much prior influence. The stop-gradient variant (99.51%) loses the gradient path but keeps most of the gain over vanilla (98.11%), in line with Section 5.3. The shifted carry, which sends block b's graph to block $b + 1$ instead of to itself, reaches 98.94%, above vanilla but well below the default, so carrying each block's graph to itself is the more effective design.

5.10 Oversmoothing Diagnostics

Table 11. Patch-token Dirichlet energy E_t and effective rank after each iteration.

	Dirichlet energy			Effective rank		
Model	t=1	t=2	t=3	t=1	t=2	t=3
Vanilla	0.082	0.031	0.014	196	121	74
LoGA-logprob	0.086	0.057	0.044	198	165	142

Table 11 reports both diagnostics. At $t = 1$ the two models are comparable (Dirichlet energy 0.082 vs. 0.086; effective rank 196 vs. 198). By $t = 3$, vanilla LoopViT's energy has dropped to 0.014 and its rank to 74, whereas LoGA keeps an energy of 0.044 and a rank of 142, about three and two times larger. In this run the carried prior therefore counteracts the oversmoothing that repeated weight-tied iterations otherwise amplify, and the patch tokens stay more diverse in later iterations.

5.11 Cost

Table 12. Training cost on RTX 3060, batch 16, 224^2, PlantVillage.

Model	img/s	Peak mem.	Extra params
LoopViT (vanilla)	188-202	1.16 GB	0
LoGA-qk	176-190	1.18 GB	24
LoGA-logprob	168-181	1.20 GB	24

As Table 12 shows, LoGA-qk processes 176-190 img/s, about 6% slower than vanilla (188-202 img/s), and LoGA-logprob about 10% slower (168-181 img/s). Peak GPU memory rises by 1.7% (qk) and 3.4% (logprob). The 24 extra parameters (the γ gates) are negligible next to the 9.91 M-parameter backbone; the overhead comes from the additional softmax and element-wise operations of the carry, not from parameter count.

5.12 Discussion

The PlantVillage run of LoGA (Section 5.4) suggests three observations. (i) Weight-tied iterations specialise: the first iteration builds largely uncalibrated features, and the second does most of the refinement. (ii) The CLS-attention graph first changes strongly between iterations and then contracts across iterations as training converges; this is consistent with, though not specific evidence for, the idea that a block's

graph at iteration $t-1$ carries information about its graph at iteration t. (iii) The default entropy exit does not interact well with label smoothing: its threshold lies about 0.62 nats below the level implied by Eq. (14), so it reduces to full-depth inference unless re-calibrated. Together with the accuracy results in Table 2, these observations support further study of carrying attention graphs across loop iterations.

Limitations. Several limitations should be kept in mind. First, for each dataset the same held-out split is used for checkpoint selection and for reporting, so the reported accuracies are best-checkpoint figures that may be optimistic; an independent test split would give an unbiased estimate. Results come from a single split per dataset and three seeds (one run for vanilla LoopViT on PlantVillage), and the published numbers in Table 3 use different splits and protocols, so they are not directly comparable. Second, the PlantVillage split is made at image level, PlantVillage contains several photographs of the same leaf, and it is known to contain capture-condition shortcuts; near-100% accuracies there should not be read as field performance. Third, Cassava and Plant Pathology 2020 are imbalanced and Cassava contains label noise. Fourth, the reference models (untied ViT, ResNet-50, DeiT-S) were trained with a shared recipe tuned for the looped models and not tuned individually. Fifth, the log-prob form materialises T_{tok}^2 maps; at higher resolution or with smaller patches, the qk form or detached carries become necessary. Sixth, Δ_t uses only the CLS row, and graphs of patch queries may keep changing after the CLS row has settled; the learned gate values, the remaining ablations of the carry design (including a shifted carry $b \to b+1$; the stop-gradient ablation is reported in Section 5.3), a full evaluation of the exit rules on the accuracy-compute plane, and a comparison of graph dynamics with vanilla LoopViT are not yet included, and the graph threshold ϵ_g requires calibration.

6 Conclusion

We introduced LoGA, a looped vision transformer for fine-grained plant disease classification built on Loop-Carried Graph Attention, which lets each weight-tied block of a looped Vision Transformer reuse its own attention graph from the previous iteration as a tempered prior. The log-prob form is equivalent to attending with a geometrically discounted sum of past logits where the floor is inactive, has bounded influence through a floor, and starts exactly at vanilla LoopViT. The qk form is exact and compatible with fused kernels. LoGA adds 24 parameters. We also examined a parameter-free graph-stability signal and showed, analytically and empirically, that a low entropy-exit threshold is not reached under label smoothing.

Trained from scratch under a common protocol, LoGA-logprob improved best-checkpoint top-1 accuracy over vanilla LoopViT on all four datasets, by 1.67 to 14.2 points, with the qk variant close behind. A 98-epoch PlantVillage run showed that the CLS-attention graph of LoGA contracts across loop iterations (Δ_t from 5.03 to 0.004 nats) and that accuracy is maintained up to twice the training depth. We see these results as encouraging but preliminary, given the single splits, the shared split for model selection and reporting, the from-scratch setting and the limitations listed above.

Future work includes evaluation on independent test splits with class-balanced metrics, repeating the gate, ablation, oversmoothing and exit analyses on the three field datasets, quantitative localisation studies with lesion-level annotations, carrying patch-level graphs through low-rank or sparse states to remove the T_{tok}^2 memory cost, combining LoGA with ImageNet-pre-trained looped backbones, and deploying compact LoGA models in the PlantDoc AI mobile application.

Data Availability Statement

The LoGA implementation, including model definitions, training and analysis scripts, configuration files and unit tests, is publicly available at https://github.com/shubham10divakar/LoGA-Loop-Carried-Graph-Attention-for-Loop-Vision-Transformers. PlantVillage, PlantDoc, Cassava Leaf Disease and Plant Pathology 2020 are publicly available from their original providers.

Funding

This work was supported without any funding.

Conflicts of Interest

The authors declare no conflicts of interest.

AI Use Statement

During the preparation of this manuscript, the authors used ChatGPT (GPT-5, OpenAI) solely for language editing, i.e. to improve the grammar, wording

and readability of text written by the authors. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the content of this publication.

Ethical Approval and Consent to Participate

Not applicable. This study uses only publicly available images of plant leaves and involves no human participants or animals.

References

[1] Strange, R. N., & Scott, P. R. (2005). Plant disease: A threat to global food security. *Annual Review of Phytopathology, 43*(1), 83-116. [CrossRef]

[2] Dosovitskiy, A. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. [CrossRef]

[3] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021, July). Training data-efficient image transformers & distillation through attention. In *International conference on machine learning* (pp. 10347-10357). PMLR. https://proceedings.mlr.press/v139/touvron21a.html

[4] Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science, 7*, 1419. [CrossRef]

[5] Singh, D., Jain, N., Jain, P., Kayal, P., Kumawat, S., & Batra, N. (2020). PlantDoc: A dataset for visual plant disease detection. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD* (pp. 249-253). [CrossRef]

[6] Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, Ł. (2018). Universal transformers. *arXiv preprint arXiv:1807.03819*. [CrossRef]

[7] Giannou, A., Rajput, S., Sohn, J. Y., Lee, K., Lee, J. D., & Papailiopoulos, D. (2023, July). Looped transformers as programmable computers. In *International Conference on Machine Learning* (pp. 11398-11442). PMLR. https://proceedings.mlr.press/v202/giannou23a.html

[8] Yang, L., Lee, K., Nowak, R., & Papailiopoulos, D. (2024, May). Looped transformers are better at learning learning algorithms. In *International conference on learning representations* (Vol. 2024, pp. 42195-42214).

[9] Shu, W. J., Qiu, X., Zhu, R. J., Chen, H. H., Liu, Y., & Yang, H. (2026). Loopvit: Scaling visual arc with looped transformers. *arXiv preprint arXiv:2602.02156*. [CrossRef]

[10] He, R., Ravula, A., Kanagal, B., & Ainslie, J. (2021, August). Realformer: Transformer likes residual attention. In *Findings of the association for computational linguistics: ACL-IJCNLP 2021* (pp. 929-943). [CrossRef]

[11] Wang, Y., Yang, Y., Bai, J., Zhang, M., Bai, J., Yu, J., ... & Tong, Y. (2021, July). Evolving attention with residual convolutions. In *International Conference on Machine Learning* (pp. 10971-10980). PMLR. https://proceedings.mlr.press/v139/wang21ab.html

[12] Xiao, T., Li, Y., Zhu, J., Yu, Z., & Liu, T. (2019). *Sharing attention weights for fast transformer*. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI) (pp. 5292-5298). [CrossRef]

[13] Venkataramanan, S., Ghodrati, A., Asano, Y., Porikli, F., & Habibian, A. (2024, May). Skip-attention: Improving vision transformers by paying less attention. In *International Conference on Learning Representations* (Vol. 2024, pp. 29654-29672).

[14] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021, October). Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF international conference on computer vision (ICCV)* (pp. 9992-10002). IEEE. [CrossRef]

[15] Shen, Z., Liu, Z., & Xing, E. (2022, October). Sliced recursive transformer. In *European Conference on Computer Vision* (pp. 727-744). Cham: Springer Nature Switzerland. [CrossRef]

[16] Zhang, J., Peng, H., Wu, K., Liu, M., Xiao, B., Fu, J., & Yuan, L. (2022, June). MiniViT: Compressing vision transformers with weight multiplexing. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 12135-12144). IEEE. [CrossRef]

[17] Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep equilibrium models. In *Advances in Neural Information Processing Systems, 32* (NeurIPS).

[18] Dong, Y., Cordonnier, J. B., & Loukas, A. (2021, July). Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International conference on machine learning* (pp. 2793-2803). PMLR. https://proceedings.mlr.press/v139/dong21a.html

[19] Wang, P., Zheng, W., Chen, T., & Wang, Z. (2022). Anti-oversmoothing in deep vision transformers via the fourier domain analysis: From theory to practice. *arXiv preprint arXiv:2203.05962*. [CrossRef]

[20] Teerapittayanon, S., McDanel, B., & Kung, H. T. (2016, December). Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)* (pp. 2464-2469). IEEE. [CrossRef]

[21] Bakhtiarnia, A., Zhang, Q., & Iosifidis, A. (2022). Single-layer vision transformers for more accurate early exits with less overhead. *Neural Networks, 153*, 461-473. [CrossRef]

[22] Banino, A., Balaguer, J., & Blundell, C. (2021). Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*. [CrossRef]

[23] Yin, H., Vahdat, A., Alvarez, J. M., Mallya, A., Kautz, J., & Molchanov, P. (2022, June). A-vit: Adaptive tokens for efficient vision transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition*

(*CVPR*) (pp. 10799-10808). IEEE. [CrossRef]

[24] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2818-2826). [CrossRef]

[25] Hughes, D., & Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060*. [CrossRef]

[26] Salman, Z., Muhammad, A., & Han, D. (2025). Plant disease classification in the wild using vision transformers and mixture of experts. *Frontiers in Plant Science, 16*, 1522985. [CrossRef]

[27] Mwebaze, E., Gebru, T., Frome, A., Nsumba, S., & Tusubira, J. (2019). iCassava 2019 fine-grained visual categorization challenge. *arXiv preprint arXiv:1908.02900*. [CrossRef]

[28] Makerere University AI Lab. (2020). Cassava leaf disease classification [Data set]. Kaggle. Retrieved from https://www.kaggle.com/competitions/cassava-leaf-disease-classification

[29] Thapa, R., Zhang, K., Snavely, N., Belongie, S., & Khan, A. (2020). The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. *Applications in plant sciences, 8*(9), e11390. [CrossRef]

[30] Borhani, Y., Khoramdel, J., & Najafi, E. (2022). A deep learning based approach for automated plant disease classification using vision transformer. *Scientific Reports, 12*(1), 11554. [CrossRef]

[31] Barman, U., Sarma, P., Rahman, M., Deka, V., Lahkar, S., Sharma, V., & Saikia, M. J. (2024). ViT-SmartAgri: Vision transformer and smartphone-based plant disease detection for smart agriculture. *Agronomy, 14*(2), 327. [CrossRef]

[32] Li, G., Wang, Y., Zhao, Q., Yuan, P., & Chang, B. (2023). PMVT: a lightweight vision transformer for plant disease identification on mobile devices. *Frontiers in Plant Science, 14*, 1256773. [CrossRef]

[33] Zhang, S., & Liu, R. (2026). Multi-scale feature fusion-based vision mamba for robust plant disease image classification on field-acquired plantdoc data. *Frontiers in Plant Science, 17*, 1842426. [CrossRef]

[34] Jain, A. K., & Nain, N. (2026). Detection and classification of cassava diseases using concatenate model. In *Real-World Applications of Machine Learning* (pp. 355-365). Singapore: Springer Nature Singapore. [CrossRef]

[35] Saood, S., Khan, S. H., & Iqbal, R. (2026). A customized Swin Transformer-based framework for cassava leaf disease classification. *Spectrum of Engineering Sciences, 4*(2), 37-48. https://thesesjournal.com/index.php/1/article/view/1950

[36] Kaur, K., & Kaur, H. (2026). PomaNet: An Optimized CNN Framework for Apple Leaf Disease Detection Using Advanced Image Processing. *Applied Fruit Science, 68*(3), 158. [CrossRef]

[37] Lee, S. H., Lee, S., & Song, B. C. (2021). Vision transformer for small-size datasets. *arXiv preprint arXiv:2112.13492*. [CrossRef]

[38] Steiner, A., Kolesnikov, A., Zhai, X., Wightman, R., Uszkoreit, J., & Beyer, L. (2021). How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*. [CrossRef]

[39] Divakar, S., & Priyadarshini, R. (2024, December). Recursive Additive Attention Based Convolutional Neural Network Model for Plant Disease Detection. In *2024 IEEE 21st India Council International Conference* (*INDICON*) (pp. 01-06). IEEE. [CrossRef]

[40] Divakar, S., Bhattacharjee, A., & Priyadarshini, R. (2021, April). Smote-DL: a deep learning based plant disease detection method. In *2021 6th international conference for convergence in technology* (*i2ct*) (pp. 1-6). IEEE. [CrossRef]

[41] Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016, September). Deep networks with stochastic depth. In *European conference on computer vision* (pp. 646-661). Cham: Springer International Publishing. [CrossRef]

[42] Shazeer, N. (2020). Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*. [CrossRef]

[43] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., & Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing, 568*, 127063. [CrossRef]

[44] Heo, B., Park, S., Han, D., & Yun, S. (2024, September). Rotary position embedding for vision transformer. In *European Conference on Computer Vision* (pp. 289-305). Cham: Springer Nature Switzerland. [CrossRef]

[45] Roy, O., & Vetterli, M. (2007, September). The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference* (pp. 606-610). IEEE. https://ieeexplore.ieee.org/abstract/document/7098875

[46] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778). [CrossRef]

[47] Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*. [CrossRef]

[48] Loshchilov, I., & Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*. [CrossRef]

[49] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision* (pp. 618-626). [CrossRef]

Subham Divakar received the B.Tech. degree in Electronics and Telecommunication Engineering from C. V. Raman Global University, Bhubaneswar, Odisha 752054, India, in 2020. He is with the Department of Electronics and Telecommunications, C. V. Raman Global University. His research interests include attention mechanisms, looped and recurrent-depth transformers, vision transformers, deep learning for plant disease recognition, and computer vision. (Email: shubham.divakar@gmail.com)

Rojalina Priyadarshini received the Ph.D. degree in Computer Science and Engineering from KIIT University, Bhubaneswar, India, in 2020. She is with the Department of Computer Science, C. V. Raman Global University, Bhubaneswar, Odisha 752054, India. Her research interests include machine learning, natural language processing, and image processing. (Email: priyadarshini.rojalina@gmail.com)