A Rescue Ladder for Getting the Columnar Architecture to Handle Split-CIFAR10 Non-Terribly
An ASCII LaTeX Protocol for Human and LLM Consumption

1 Purpose

This document gives a detailed rescue protocol for the current Split-CIFAR10 columnar experiments. The immediate goal is *not* to solve continual learning on CIFAR10 right away. The immediate goal is more basic and more important:

> make the columnar architecture learn the current CIFAR10 task non-terribly before studying continual learning.

The protocol is designed for both human and LLM use. It therefore includes:

- a diagnosis of the current failed runs,
- a conceptual explanation of what is probably going wrong,
- an 8-step rescue ladder with explicit architectures and hyperparameters,
- success criteria and failure interpretations for each step,
- and a reintroduction schedule for more advanced mechanisms only after the base architecture is working.

The core philosophy of the rescue ladder is:

1. separate visual representation learning from continual-learning control,
2. separate simple columnar computation from sparse support selection,
3. separate ordinary supervised optimization from predictive-coding-style training,
4. and only reintroduce advanced modular and causal machinery after the current-task backbone is proven competent.

2 Executive diagnosis of the inspected runs

Three inspected run archives reveal a very consistent picture. Two of them are still effectively smoke tests, and the more serious one is large enough to be informative but still does not establish a competent CIFAR10 current-task learner.

2.1 Run family A: debug-scale smoke tests

The archives dated `20260421` and `20260423` have the following properties:

- `epochs_per_task = 1`
- `fast_dev_max_train_batches = 2`
- `fast_dev_max_test_batches = 2`
- `batch_size = 32`
- `patch_embed_dim = 8`
- `memory_dim = 16`
- shell sizes of only `[2,4,2]`
- training mode `pc`
- weight decay `0.1`

This means each task is effectively being trained on only about 64 training examples and evaluated on only about 64 test examples. Such runs are useful for plumbing checks, not for drawing conclusions about CIFAR learning. Their poor accuracy matrices therefore do *not* establish that the columnar architecture cannot handle CIFAR10. They establish only that the debug-scale setting is far too weak to be informative.

2.2 Run family B: the informative but still failing run

The archive dated `20260422` is more serious:

- `epochs_per_task = 3`
- no fast-dev caps
- `batch_size = 256`
- `patch_embed_dim = 64`
- `memory_dim = 32`
- `num_columns = 40`
- `shared_columns = 4`
- `topk_nonshared = 5`
- shell sizes `[8,16,8]`
- training mode `pc`
- weight decay `0.1`

This run is therefore large enough to say something real. What it says is not that continual learning has failed in some subtle way, but that the architecture is still not reliably learning the *current task* well enough. The task-diagonal accuracies after finishing each task are approximately:

$$0.618,\ 0.6955,\ 0.5915,\ 0.822,\ 0.793.$$

The final row after the fifth task is approximately:

$$0.664,\ 0.500,\ 0.500,\ 0.500,\ 0.793.$$

These numbers imply three things at once:

1. some current tasks are underlearned even before worrying about forgetting,
2. later sequential training then drives multiple earlier tasks back to chance,
3. and the whole system is therefore not yet in a regime where continual-learning-specific conclusions are trustworthy.

3 Interpretation of the specific diagnosis

The diagnosis to explain is:

> underlearning on some current tasks, especially task 0 and 2;
> forgetting / collapse to chance on tasks 1, 2, and 3 after later training;
> still no real CIFAR visual stem, just patch embedding;
> PC training mode being used before verifying that the architecture can learn CIFAR well under ordinary backprop;
> high weight_decay = 0.1, which may be too harsh for an already underpowered CIFAR representation.

Each part has a precise architectural meaning.

3.1 1. Underlearning on some current tasks

When the current-task diagonal accuracy on a 2-class CIFAR10 split remains only around 0.59 to 0.70 after training, that is not a subtle continual-learning phenomenon. That is a sign that the system has not built a competent representation for the current task.

This matters because continual learning should only be analyzed once the system can answer the simpler question:

> Can the architecture learn the present binary task well when that task is the one currently being optimized?

If the answer is already "not very well," then later forgetting statistics are not especially informative. A model that never learned a task strongly cannot "forget" it in the interesting sense.

3.2 2. Forgetting / collapse to chance later

The final row showing multiple tasks at 0.50 means several earlier tasks end at chance. This certainly indicates forgetting, but the forgetting is not interpretable in isolation because the early task learning itself was weak. The sequential system is therefore failing in two distinct ways:

- weak current-task competence,
- and destructive later interference.

The rescue protocol must therefore first attack weak current-task competence. Only after that should it address interference.

3.3 3. No real CIFAR visual stem

A patch embedding alone is often sufficient for very simple grayscale or low-complexity toy domains. CIFAR10 is not such a domain. It contains color, texture, background clutter, small-object variability, and significantly richer nuisance variation than MNIST.

When CIFAR10 is fed straight into a small patch embedding and then immediately fragmented into sparse columns, the architecture is forcing the columnar modules to solve low-level vision, medium-level feature composition, and task-level modularity simultaneously. That is too much to ask of a small unstable system.

A genuine CIFAR rescue path therefore requires a real shared visual stem before the columnar machinery.

3.4 4. PC mode used too early

Predictive-coding-style or PC-like training may eventually be valuable, but it is an additional optimization difficulty. If it is used before ordinary supervised learning has established that the architecture can learn CIFAR at all, then one cannot tell whether failure comes from:

- representational weakness,
- optimization instability,
- or the interaction of both.

Therefore the correct scientific order is:

1. ordinary supervised backprop first,
2. then columnar structure,
3. then sparse support,
4. then sequential training,
5. only then PC and more advanced causal-control additions.

3.5 5. Weight decay 0.1 is likely too harsh

A weight decay of 0.1 can be sensible in some large-scale or highly overparameterized settings with robust feature learners. In the present CIFAR10 columnar setting it is likely too aggressive.

Why? Because the model is already weak in three ways:

- the front end is underpowered,
- the architecture is heavily fragmented,
- and the training dynamics are already more difficult than standard CNN training.

Large weight decay in this situation pushes the model further toward underfitting. It suppresses the formation of the very intermediate visual filters and feature couplings that the model still urgently needs.

A much safer initial range for the rescue ladder is:

$$10^{-4} \text{ to } 5 \times 10^{-4}$$

for most of the early steps.

4 The big conceptual mistake to avoid

The main mistake would be to continue asking the architecture to solve:

- low-level CIFAR vision,
- modular sparse routing,
- shell-tier internal specialization,
- task-sequential non-forgetting,
- and PC-style optimization

all at once.

The rescue ladder exists to unbundle these. At each stage, only one or two sources of difficulty should be active. The question at each stage is:

> What exactly is the next thing the architecture must prove it can do?

5 Global rules for the entire rescue ladder

These rules apply to all eight stages unless explicitly overridden.

5.1 Dataset and task protocol

Use Split-CIFAR10 with 5 binary tasks:

$$(0,1),\ (2,3),\ (4,5),\ (6,7),\ (8,9).$$

For single-task stages, start with task 0, i.e. CIFAR10 classes 0 vs 1. Only move to later tasks after the architecture handles one binary task well.

5.2 Data preprocessing

For the first rescue stages, use only light standard augmentation:

- random crop with padding 4,
- random horizontal flip,
- standard CIFAR10 normalization,
- no mixup,
- no cutmix,
- no RandAugment,
- no strong contrastive augmentations.

The point is diagnostic clarity, not leaderboard performance.

Use CIFAR10 normalization:

$$\mu = (0.4914, 0.4822, 0.4465), \qquad \sigma = (0.2470, 0.2435, 0.2616).$$

5.3 Common optimizer defaults

Unless a step says otherwise, use:

- optimizer: AdamW,
- learning rate: 3×10^{-4},
- weight decay: 5×10^{-4},
- gradient clipping: 1.0,
- batch size: 128,
- cosine or flat learning-rate schedule with 5-epoch warmup if training is long.

5.4 Global "off" switches for early stages

For Steps 1 through 6, set the following to off unless explicitly stated otherwise:

- support selector learning,
- replay-bank support proposals,
- TransWeave,
- route certificates as decision authority,
- per-weight causal/SB corrections,
- shell demotion,
- support bank,
- PC training mode,
- EWC,
- exact support teacher.

Those are not "bad" mechanisms. They are just not the first thing to test.

5.5 Global success convention

For any step involving a single binary CIFAR task, the stage is considered passed if:

1. train accuracy rises clearly above chance,
2. test accuracy is stably above 80%,
3. train and test losses both fall materially below chance-loss,
4. and learning curves are smooth rather than degenerate or wildly unstable.

A stronger soft target is 85% or better test accuracy.
For multi-task joint-training stages, the stage is considered passed if:

1. the average task accuracy is comfortably above chance,
2. no task collapses to near chance,
3. and the diagonal task accuracies are not the only respectable numbers.

6 The eight-step rescue ladder

Summary table

Step	Purpose	Architecture	Training mode	Pass criterion
1	Data and pipeline sanity	Plain CNN baseline	Backprop	$> 85\%$ on one binary task
2	Real CIFAR stem + one columnar trunk	Shared stem + 1 active column, no shells	Backprop	$> 80\%$ on one binary task
3	Core columnar capacity upper bound	Same as Step 2, but 5-task joint training	Backprop	joint average clearly $>$ chance
4	Fragmentation test	Multi-column, all active, no shells	Backprop	small drop from Step 2
5	Sparse support cost test	Multi-column, fixed sparse support, no selector	Backprop	$> 78\%$ on one binary task
6	Shells without CL	Sparse support + shells, but no shell dynamics, joint training	Backprop	multi-task competence retained
7	Sequential learning with oracle support	Sequential task-incremental, fixed per-task support	Backprop	diagonals good on every task
8	Re-enter PC carefully	Same as Step 7, but switch to PC or hybrid training	PC or hybrid	no catastrophic drop relative to Step 7

7 Step 1: plain CNN sanity baseline

7.1 Question answered

Is there any data, label, preprocessing, or general training plumbing bug that would make CIFAR10 fail even before the columnar architecture enters the picture?

7.2 Architecture

Use a completely ordinary small CNN on a single binary task, e.g. task 0 = classes 0 vs 1.

Recommended architecture:

$$
\begin{aligned}
&\mathrm{Conv}(3, 64, 3 \times 3, \text{stride } 1),\ \mathrm{ReLU},\\
&\mathrm{Conv}(64, 64, 3 \times 3, \text{stride } 1),\ \mathrm{ReLU},\ \mathrm{MaxPool}(2),\\
&\mathrm{Conv}(64, 128, 3 \times 3, \text{stride } 1),\ \mathrm{ReLU},\\
&\mathrm{Conv}(128, 128, 3 \times 3, \text{stride } 1),\ \mathrm{ReLU},\ \mathrm{MaxPool}(2),\\
&\mathrm{Conv}(128, 256, 3 \times 3, \text{stride } 1),\ \mathrm{ReLU},\\
&\mathrm{GlobalAvgPool},\\
&\mathrm{Linear}(256, 2).
\end{aligned}
$$

Use BatchNorm only if desired, but keep the baseline simple.

7.3 Training

- Task: one binary split only
- Epochs: 30
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 5×10^{-4}
- Training mode: ordinary backprop with cross-entropy

7.4 Pass criterion

Test accuracy should exceed 85% on the binary task, ideally by a comfortable margin. If this fails, *stop immediately* and debug:

- class filtering,
- label remapping,
- train/test split,
- normalization,
- augmentation,
- and optimizer plumbing.

7.5 Why this step matters

If this baseline fails, then the issue is not "columnar CIFAR." It is the data pipeline or generic training setup.

8 Step 2: shared CIFAR stem + one active columnar trunk, no shells

8.1 Question answered

> Can the core columnar computational style, without fragmentation or shell complications, learn one binary CIFAR task when given a real visual stem?

8.2 Architecture

Use a shared CIFAR stem first, then a single active columnar trunk.

Recommended shared stem:

$$\begin{aligned}&\mathrm{Conv}(3, 32, 3 \times 3, 1),\ \mathrm{ReLU},\\ &\mathrm{Conv}(32, 64, 3 \times 3, 2),\ \mathrm{ReLU},\\ &\mathrm{Conv}(64, 64, 3 \times 3, 1),\ \mathrm{ReLU}.\end{aligned}$$

This gives a meaningful visual front end before the columnar block.

Recommended columnar trunk:

- total columns: 1 active column only
- microcolumns: K, L, B present structurally
- memory dimension: 64
- shell sizes: $[0, 0, 0]$, i.e. shells disabled
- composer: disabled
- head: 2-way head for the single task

8.3 Training

- Task: one binary split only
- Epochs: 30
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}
- Training mode: backprop with cross-entropy

8.4 Pass criterion

Test accuracy should exceed 80%. The soft target is 85%.

8.5 Failure interpretation

If Step 1 passes but Step 2 fails, the problem is no longer generic CIFAR plumbing. It is something about the columnar block computation itself, even before sparse support and shells.

8.6 If Step 2 fails

Try only one change at a time:

a. increase memory dimension from 64 to 96,

b. or add one more stem convolution: $\mathrm{Conv}(64, 96, 3 \times 3, 2)$,

c. but do not yet add shells or selectors.

9 Step 3: same one-column architecture, but joint training across all 5 tasks

9.1 Question answered

> What is the non-continual-learning upper bound of the minimal columnar backbone on the full Split-CIFAR10 problem if all tasks are trained jointly?

9.2 Architecture

Use exactly the Step 2 architecture, but attach task-local 2-way heads for all 5 tasks.

9.3 Training

- Data: all 5 binary tasks mixed jointly during training
- Heads: task-local 2-way heads
- Epochs: 40
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}

9.4 Pass criterion

The average task accuracy should be comfortably above chance and no task should sit near 50%. A practical target is average task accuracy at or above 75%.

9.5 Failure interpretation

If Step 3 fails, the minimal single-trunk columnar backbone is still not representationally strong enough for CIFAR10, even when forgetting is removed from the problem. That means do *not* proceed to support selection or CL. Strengthen the trunk first.

10 Step 4: multi-column, all active, no shells

10.1 Question answered

> Can multiple columns coexist without immediately destroying CIFAR performance when all columns are active and the selector is removed from the equation?

10.2 Architecture

Use the same shared stem as Step 2, but replace the single trunk with a small multi-column system.

Recommended settings:

- total columns: 8
- shared columns: 2
- adaptive columns: 6
- active support: all 8 active for every example
- memory dimension per column: 48
- shell sizes: $[0, 0, 0]$
- composer: simple learned gated sum with hidden dimension 64
- attention composer: off

10.3 Training

- Task: one binary split only
- Epochs: 30
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}

10.4 Pass criterion

Performance should stay near Step 2, ideally within 3 to 5 percentage points. So if Step 2 achieved 84%, Step 4 should still be around 79% or better.

10.5 Failure interpretation

If Step 2 works but Step 4 fails badly, then the main problem is architectural fragmentation or poor aggregation among columns. Not continual learning. Not the selector. Just fragmentation.

11 Step 5: multi-column with fixed sparse support, no selector

11.1 Question answered

How much does hard sparsity itself cost before learned support selection is introduced?

11.2 Architecture

Use a slightly larger pool but a fixed support mask.

Recommended settings:

- total columns: 12
- shared columns: 3
- adaptive columns: 9
- reserve columns: 0
- active support per example: 3 shared + 4 fixed nonshared = 7 total
- memory dimension per column: 48
- shell sizes: $[0, 0, 0]$
- selector: off
- fixed support identity for the task: use the same nonshared 4 for all examples in this stage

11.3 Training

- Task: one binary split only
- Epochs: 30
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}

11.4 Pass criterion

Test accuracy should exceed 78%. The performance drop from Step 4 should be modest rather than catastrophic.

11.5 Failure interpretation

If all-active columns work but fixed sparse support fails badly, then sparse masking is being applied before each active column has enough individual capacity. Possible fixes are:

a. increase per-column memory dimension from 48 to 64,

b. reduce total columns from 12 to 10,

c. or increase active nonshared support from 4 to 5.

12 Step 6: sparse support plus shells, but no shell dynamics, joint training

12.1 Question answered

> Can the system retain CIFAR competence when shells are present structurally, but pruning, promotion, and demotion are still disabled?

12.2 Architecture

Now reintroduce shells, but only as static capacity, not as dynamic self-editing machinery.

Recommended settings:

- total columns: 16
- shared columns: 4
- adaptive columns: 12
- reserve columns: 0
- active support per task: 4 shared + 4 fixed nonshared = 8 total
- memory dimension per column: 48
- shell sizes: $[4, 8, 12]$
- shell inhibition strengths: $\gamma = [0.20, 0.12, 0.06]$
- pruning: off
- promotion: off
- demotion: off

12.3 Training

- Data: all 5 tasks jointly
- Heads: task-local 2-way heads
- Epochs: 50
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}

12.4 Pass criterion

Average joint-task performance should remain respectable, and no task should collapse to chance. A practical target is average task accuracy at or above 75%.

12.5 Failure interpretation

If Step 5 works but Step 6 fails, then shells are hurting the model even before dynamic shell control is introduced. The fix is not more shell machinery. The fix is simpler shells, weaker inhibition, or larger core width. Try:

- shells $[3, 6, 9]$ instead of $[4, 8, 12]$,
- or memory dimension 64 instead of 48.

13 Step 7: sequential task-incremental training with oracle or fixed support, still backprop

13.1 Question answered

> Can the architecture perform non-trivial sequential learning on CIFAR10 when support assignment is not learned, but instead fixed or oracle-specified?

13.2 Architecture

This is the first true continual-learning step, but support selection is still simplified.

Recommended settings:

- total columns: 24
- shared columns: 4

- adaptive columns: 20
- reserve columns: 0
- per-task support: 4 shared + 4 fixed nonshared = 8 total
- use disjoint nonshared supports across the 5 tasks for maximum diagnostic clarity
- memory dimension per column: 64
- shell sizes: $[4, 8, 12]$
- pruning: off
- promotion: off
- demotion: off
- selector: off

A simple deterministic support allocation is:

$$\begin{aligned}
&\text{task 0: columns } \{4, 5, 6, 7\},\\
&\text{task 1: columns } \{8, 9, 10, 11\},\\
&\text{task 2: columns } \{12, 13, 14, 15\},\\
&\text{task 3: columns } \{16, 17, 18, 19\},\\
&\text{task 4: columns } \{20, 21, 22, 23\}.
\end{aligned}$$

13.3 Training

- Training: task-incremental, sequential
- Epochs per task: 20
- Batch size: 128
- Optimizer: AdamW
- Learning rate: 3×10^{-4}
- Weight decay: 1×10^{-4}
- Heads: task-local 2-way heads

13.4 Pass criterion

For each task, the current-task diagonal accuracy at the end of that task should exceed 75%. If some tasks never exceed 65% even with oracle support, do not proceed to learned support.

13.5 Why this step matters

This stage separates two issues:

- can the architecture do sequential learning at all when modularity is handed to it,
- versus can the selector discover that modularity.

If Step 7 fails, learned selection is not the immediate problem. The base sequential architecture is still not ready.

14 Step 8: re-enter PC or hybrid training only after Step 7 passes

14.1 Question answered

> Can PC-style training be introduced *after* the backprop-based columnar CIFAR system is already competent, without causing a large collapse?

14.2 Architecture

Use exactly the Step 7 architecture at first. Do not change the network size and the training mode simultaneously. That would confound the result.

Thus keep:

- total columns: 24
- shared columns: 4
- adaptive columns: 20
- fixed per-task supports as in Step 7
- memory dimension: 64
- shell sizes: $[4, 8, 12]$

14.3 Training mode schedule

Do this in two subphases.

Step 8A: Hybrid warm re-entry

- initialize from a successful Step 7 checkpoint
- train with mixed loss: 75% ordinary CE / 25% PC-style loss for the first experiments
- epochs per task: 15
- learning rate: 1.5×10^{-4}
- weight decay: 1×10^{-4}

Step 8B: Pure PC only if 8A is stable

- same architecture
- full PC training mode only after the hybrid version remains close to Step 7
- learning rate: 1×10^{-4}
- weight decay: 5×10^{-5}

14.4 Pass criterion

The transition from Step 7 to Step 8A should not produce a large current-task accuracy collapse. A practical criterion is:

> no more than a 5 percentage point drop in diagonal current-task accuracy relative to Step 7.

Only if this holds should pure PC be attempted.

14.5 Failure interpretation

If Step 7 passes and Step 8 fails badly, the conclusion is not "columnar CIFAR failed." The conclusion is:

> PC-style optimization is not yet mature enough for this CIFAR columnar regime.

That is an important and scientifically useful conclusion.

15 What stays off until the ladder is passed

Even after Step 8, do *not* immediately turn everything back on. Only after the ladder is passed should one gradually reintroduce:

- learned support selectors,
- exact boundary teachers,
- replay-bank support proposals,
- route certificates as decision authority,
- per-weight causal/SB corrections,
- TransWeave,
- pruning,
- promotion,

- demotion,
- composer transfer,
- and symbolic or PLN sidecars.

The order should still be incremental.

16 Recommended parameter fallbacks if a stage fails

The first fallback changes should be conservative and one-at-a-time.

16.1 If representation is too weak

Increase in this order:

1. memory dimension by +16 or +32,
2. then shared stem width,
3. only then shell size.

Do *not* first increase the number of columns. That usually worsens fragmentation before fixing representation.

16.2 If fragmentation is too strong

Change in this order:

1. fewer total columns,
2. slightly larger active support,
3. larger per-column memory,
4. simpler composer.

16.3 If regularization is too strong

Change in this order:

1. lower weight decay,
2. remove non-essential losses,
3. simplify shell inhibition,
4. remove PC losses first, not last.

17 A recommended concrete first sequence of runs

If compute budget is limited, I would run the following exact sequence first:

1. Step 1 exactly as specified.
2. Step 2 exactly as specified.
3. Step 4 exactly as specified.
4. Step 5 exactly as specified.
5. Step 7 exactly as specified.

This sequence minimizes wasted effort by first proving:

1. the data pipeline works,
2. the one-column backbone works,
3. multiple columns can coexist,
4. sparse support is not catastrophic,
5. and sequential learning is possible with oracle support.

Only after that would I run Steps 3, 6, and 8 for upper-bound and PC-reentry diagnostics.

18 Interpretation rules for the rescue ladder

The ladder is designed so that every failure has a clean meaning.

18.1 If Step 1 fails

The issue is general CIFAR plumbing. Do not blame the columnar architecture.

18.2 If Step 1 passes and Step 2 fails

The issue is the core columnar block. Do not blame selectors, shells, or CL.

18.3 If Step 2 passes and Step 4 fails

The issue is fragmentation or aggregation. Do not blame selectors or CL.

18.4 If Step 4 passes and Step 5 fails

The issue is hard sparsity or insufficient per-column capacity. Do not blame learned support selection.

18.5 If Step 5 passes and Step 6 fails

The issue is shell capacity or shell structure. Do not turn on shell dynamics. Simplify shells first.

18.6 If Step 6 passes and Step 7 fails

The issue is sequential training under fixed modularity. Do not move to learned selectors yet.

18.7 If Step 7 passes and Step 8 fails

The issue is PC-style optimization in this regime. That is a training-mode problem, not a proof that CIFAR10 and columnar modularity are incompatible.

19 Bottom line

The currently inspected CIFAR10 runs do not justify the conclusion that the columnar architecture has fundamentally failed on CIFAR10. They justify a narrower and more actionable conclusion:

> The experiments have asked a too-small and too-unstable system to solve CIFAR vision, sparse modular routing, shell organization, sequential continual learning, and PC optimization at the same time.

The rescue ladder fixes this by enforcing the correct scientific order:

1. verify CIFAR plumbing,
2. verify CIFAR representation,
3. verify multi-column coexistence,
4. verify sparse support,
5. verify sequential learning with fixed support,
6. only then reintroduce PC and advanced control machinery.

If followed carefully, this ladder should reveal *where* the failure truly lies. That is the real objective of the next round of experiments.