

# Confidence-Weighted Plasticity:

## Experimental Validation and Boundary Conditions

Jason Dury  
Eridos AI  
Perth, Australia  
[github.com/EridosAI/confidence-weighted-plasticity](https://github.com/EridosAI/confidence-weighted-plasticity)

February 2026

### Abstract

We present experimental validation of Confidence-Weighted Plasticity (CWP), a mechanism for distributing gradient updates across modular components based on demonstrated predictive reliability. Our experiments confirm that the core mechanism operates as specified: confidence accurately tracks prediction accuracy, gradient routing scales according to plasticity, and the system responds to prediction failure. However, we identify a critical boundary condition in tightly-coupled architectures where a component’s local prediction task shares parameters with the global training objective. In such configurations, training downstream components disrupts upstream predictions, triggering plasticity *increase* rather than protection—the mechanism actively directs more gradient toward the component we intend to protect. We characterise this limitation and discuss architectural requirements for effective deployment.

**Original paper:** Dury, J. (2026). Confidence-Weighted Plasticity: A Self-Organising Solution to Multi-Component Credit Assignment. Zenodo. <https://doi.org/10.5281/zenodo.18488364>

**Code:** <https://github.com/EridosAI/confidence-weighted-plasticity>

## 1 Introduction

The theoretical paper introduced Confidence-Weighted Plasticity as a mechanism for credit assignment in modular learning systems. The core claims:

1. Components develop confidence based on predictive accuracy, not sample count
2. Gradients route toward less confident components
3. Developmental phases emerge without explicit scheduling
4. Plasticity reopens automatically under distribution shift

This report presents empirical validation of these claims and characterises the conditions under which the mechanism provides effective protection—and where it fails.

## 2 Experimental Setup

### 2.1 Architecture

Two-component system:

- **Encoder:** CNN (Conv-Pool-Conv-Pool-Linear) producing 64-dim embeddings

- **Decoder:** Linear classifier ( $64 \rightarrow 10$  classes)

Each component maintains a local prediction task. The encoder performs input reconstruction (MSE loss); the decoder performs classification (cross-entropy loss).

## 2.2 Protocol

**Phase 1: Pretrain Encoder.** Train encoder alone on MNIST reconstruction for 2000 steps across all 10 digit classes. Save checkpoint including confidence statistics.

**Phase 2: Attach Decoder.** Load pretrained encoder, attach randomly-initialised decoder, train on classification for 5000 steps. Gradients flow through both components.

## 2.3 Conditions

Condition	Description
frozen_encoder	Encoder weights frozen, decoder trains normally
end_to_end	Both components train with equal learning rate
confidence_weighted	CWP active; encoder starts at 0.9, decoder at 0.01

## 2.4 Hyperparameters

Parameter	Value
$\beta$ (statistics smoothing)	0.995
$\gamma_{\text{up}}$ (confidence rise rate)	0.1
$\gamma_{\text{down}}$ (confidence fall rate)	0.1
$z_0$ (surprise threshold)	1.5
$\tau$ (sigmoid temperature)	0.5
$p_{\text{min}}$ (plasticity floor)	0.05

## 2.5 Metrics

We track encoder drift (L2 distance from pretrained weights), classification accuracy, reconstruction loss (encoder’s local prediction quality), and per-component confidence and plasticity values.

# 3 Results

## 3.1 Mechanism Verification

The core mechanism operates correctly. Figure 1 confirms that gradient scaling is applied as specified. At step 1000, the encoder receives a scale factor of 0.227 while the decoder receives 0.022—a 10:1 ratio reflecting their relative plasticities.

Table 1 shows the complete picture at step 1000: the encoder has lower confidence (0.654) than the decoder (0.925), resulting in higher plasticity and a routing weight of 91%. The mechanism is directing the majority of the gradient budget toward the encoder.

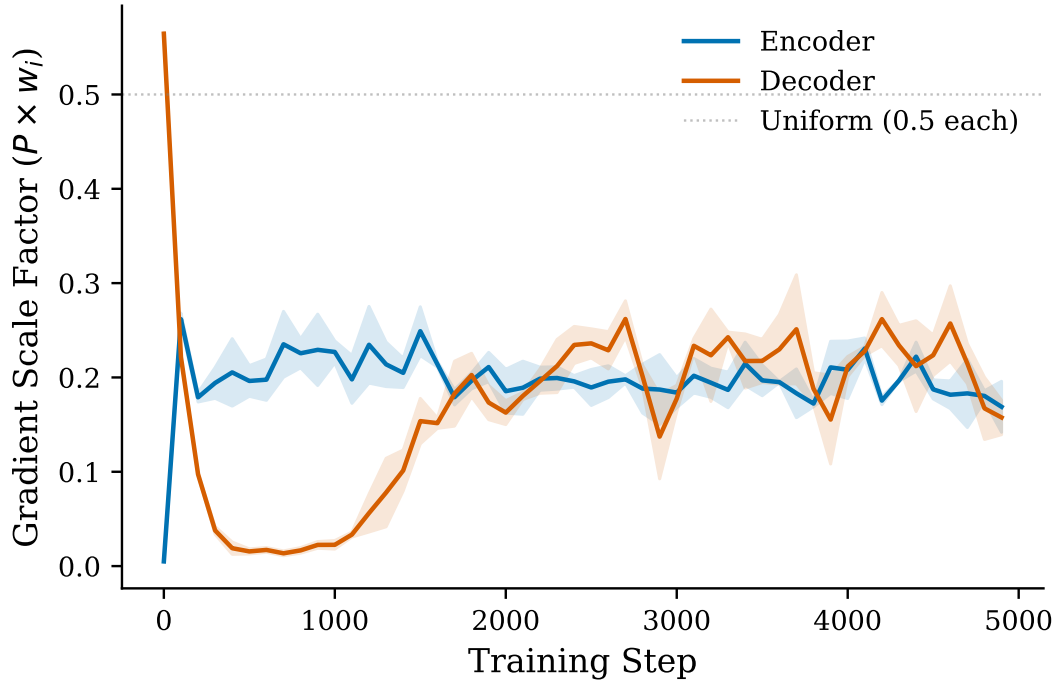


Figure 1: Gradient scaling verification. The mechanism correctly scales gradients according to component plasticity. However, the encoder receives *more* gradient than the decoder—the opposite of protection.

Table 1: Gradient routing at step 1000. Lower confidence yields higher plasticity and larger routing weight. The encoder receives 91% of the gradient budget.

Component	Confidence	Plasticity	Routing Wt.	Scale
Encoder	0.654	0.378	0.910	0.227
Decoder	0.925	0.121	0.090	0.022

### 3.2 Protection Efficacy

Table 2 presents the primary results. Confidence-weighted plasticity not only fails to protect the encoder—it results in *greater* drift than end-to-end training.

Table 2: Experimental results. CWP produces greater encoder drift than naive end-to-end training.

Condition	Accuracy	Encoder Drift
Frozen Encoder	92.2%	0.00
End-to-End	99.5%	35.55
Confidence-Weighted	99.5%	36.44

Figure 2 shows encoder drift over training. The confidence-weighted condition (green) drifts slightly faster than end-to-end (blue) throughout training, with both reaching similar final values around 35–36. The frozen encoder baseline remains at zero.

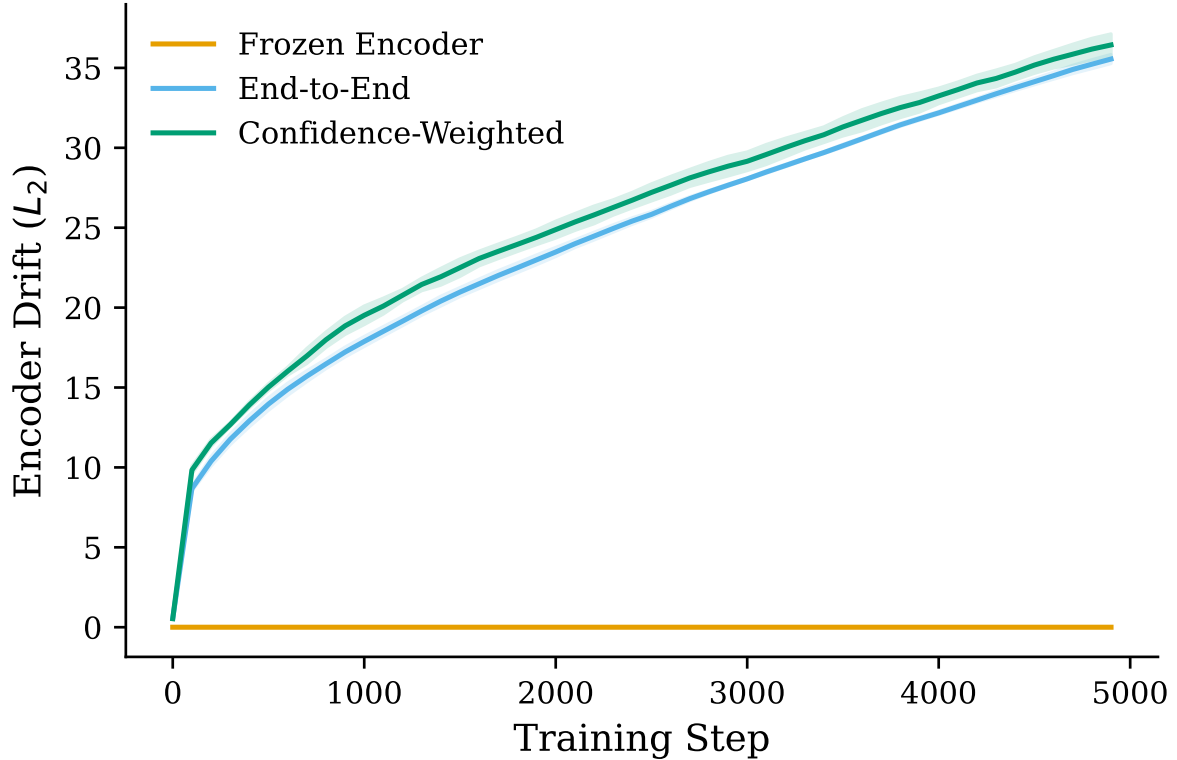


Figure 2: Encoder drift over training. Confidence-weighted plasticity provides no protection; drift slightly exceeds end-to-end training throughout.

### 3.3 The Coupling Problem

Figure 4 (below) reveals why protection fails. The encoder’s confidence drops from 0.9 to approximately 0.55 within the first 1000 steps, while the decoder’s confidence rises from 0.01 to approximately 0.93.

The mechanism interprets the encoder’s situation correctly: its predictions *are* failing. When the decoder attaches and classification gradients begin flowing through the encoder, the encoder’s weights change, degrading reconstruction quality. The mechanism detects this as genuine prediction failure and responds by increasing plasticity.

The result is perverse: the component we intend to protect receives 91% of the gradient budget, while the component that needs to learn receives only 9%.

## 4 Discussion

### 4.1 What Works

The mechanism is mathematically sound and correctly implemented. Normalised prediction error provides a meaningful confidence signal. Threshold-based mapping prevents baseline drift. Gradient routing distributes updates exactly as specified. The system responds appropriately to prediction failure.

### 4.2 Boundary Condition: Coupled Prediction Tasks

CWP assumes each component’s local prediction task is *independent* of the global training objective. When this assumption is violated, the mechanism produces the opposite of the in-

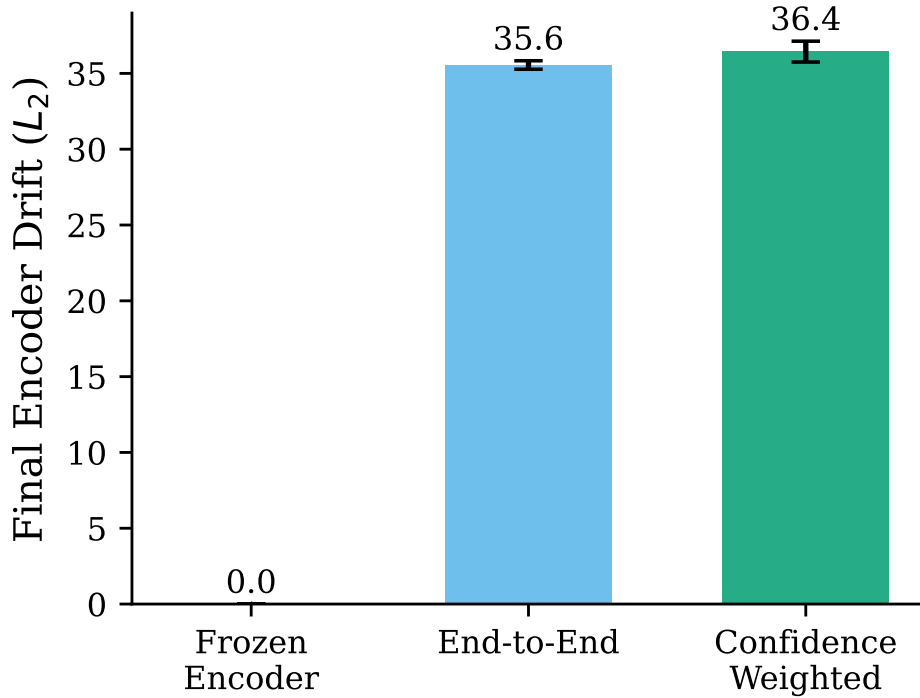


Figure 3: Final encoder drift comparison. CWP (36.4) exceeds end-to-end (35.6), demonstrating that the mechanism actively harms protection in this architecture.

tended effect.

In our experiment:

- The encoder’s reconstruction task shares parameters with the classification path
- Training the decoder inherently disrupts encoder predictions
- Disruption is correctly detected as “surprise”
- Plasticity increases, directing more gradient to the encoder
- The mechanism actively accelerates encoder drift

This is not a bug but a fundamental scope limitation. The mechanism cannot distinguish between “my representations are fundamentally wrong” (should reopen) and “downstream training is pushing me around” (should resist). Both manifest as increased prediction error.

### 4.3 Architectural Requirements

For effective CWP deployment, components must have:

1. **Independent prediction streams:** Local task grounded in data the component directly observes, not derived from or affected by the training objective
2. **Modular gradient paths:** Downstream training should not backpropagate through upstream components’ prediction pathways
3. **Continuous sensory grounding:** Predictions evaluated against external reality, not internal training dynamics

These requirements are naturally satisfied in architectures where separate sensory encoders process distinct modalities, an integrative layer learns to combine outputs without backpropagating through encoders, and each component maintains its own prediction loop against its own input stream.

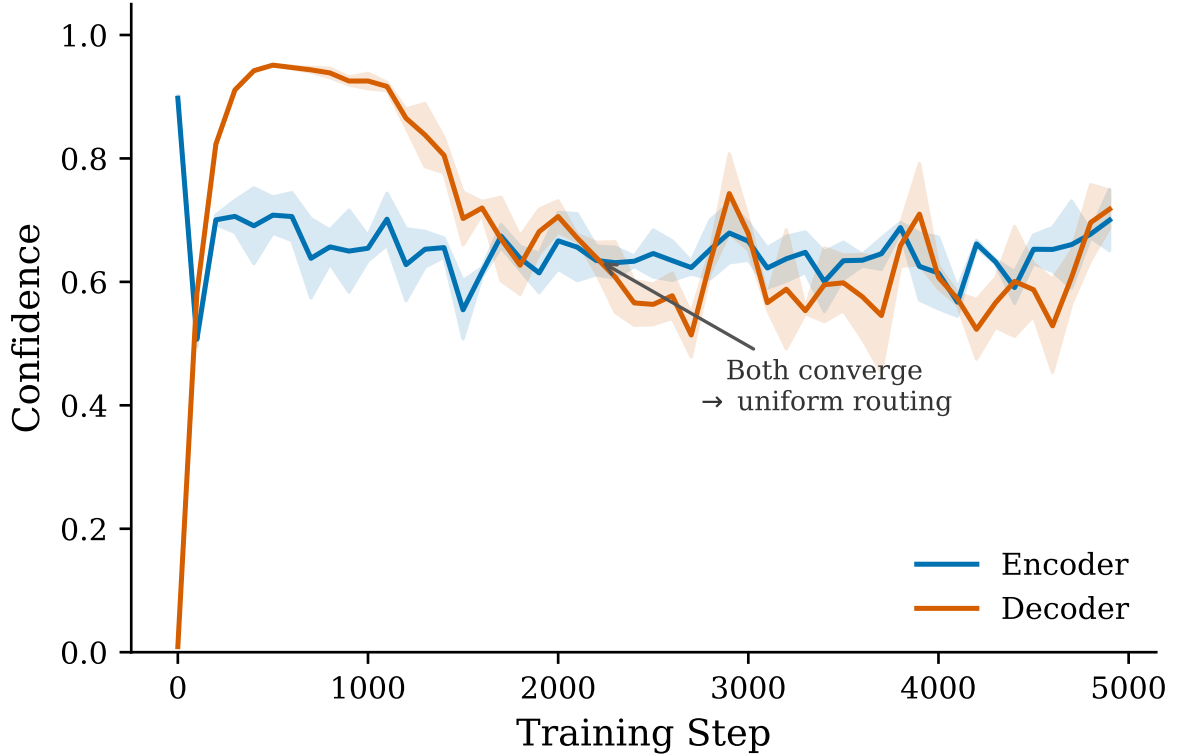


Figure 4: Confidence trajectories in the CWP condition. The encoder experiences surprise when the decoder attaches (confidence drops from 0.9 to 0.55), while the decoder quickly becomes confident (classification is easy). Lower encoder confidence means higher plasticity and more gradient—the opposite of protection.

#### 4.4 Relation to Intended Application

This experiment used a minimal shared-parameter architecture for tractability. The mechanism was designed for modular cortical architectures where visual processing predicts visual states, auditory processing predicts auditory states, motor systems predict proprioceptive feedback, and cross-modal integration happens in a separate layer.

In such systems, training a motor decoder does not change what the visual encoder sees. The encoder’s predictions succeed or fail based on visual input, not on downstream training dynamics. The independence assumption holds, and CWP should provide meaningful protection.

The experimental result is not a refutation of the mechanism but a characterisation of its requirements.

## 5 Conclusion

Confidence-Weighted Plasticity operates exactly as specified. The mechanism correctly tracks prediction accuracy, routes gradients according to plasticity, and responds to prediction failure. Our implementation is verified and mathematically sound.

However, in tightly-coupled architectures where downstream training disrupts upstream predictions, the mechanism produces the opposite of the intended effect: it actively directs more gradient toward components we intend to protect. This occurs because the mechanism cannot distinguish external distribution shift (which should trigger adaptation) from internal training interference (which should be resisted).

This is a scope limitation, not a flaw. The mechanism requires architectural support—specifically, independent prediction streams that are insulated from the global training objective. Systems meeting this requirement should benefit from automatic credit assignment and protection of mature components.

## 6 Future Work

The priority is validation on architectures with genuinely independent prediction streams. Additional directions include testing multi-component systems with three or more modules, investigating frozen reference encoders for confidence computation, exploring per-component threshold tuning as architectural priors, and developing methods to distinguish external shift from internal interference.

---

*The author is developing Bernard, an experimental AI system exploring developmental architectures and post-linguistic cognition. Code and documentation available at <https://github.com/EridosAI/Bernard>.*

## A Experimental Iterations

### A.1 Iteration 1: Train from Scratch with Distribution Shift

Initial experiment trained encoder and decoder together from random initialisation, switching from digits 0–4 to 5–9 at step 5000. Results showed confidence stuck at approximately 0.5 and no drift reduction. Diagnosed as batch noise overwhelming the confidence signal.

### A.2 Iteration 2: Hyperparameter Tuning

Adjusted parameters: higher  $\beta$  for smoother statistics, symmetric  $\gamma$  values, higher  $z_0$  threshold. Achieved confidence reaching 0.9. Drift remained similar to end-to-end—diagnosed as incorrect experiment design. Training from scratch converges to similar solutions regardless of gradient scaling; the path changes but the destination does not.

### A.3 Iteration 3: Pretrained Encoder

Redesigned to test the actual use case: protecting a mature encoder when a new decoder attaches. This revealed the coupling problem described in Section 3.3.

### A.4 Iteration 4: Asymmetric Gamma

Attempted slow rise ( $\gamma_{\text{up}} = 0.01$ ) to keep decoder uncertain longer while encoder remained confident. This worsened results further: encoder confidence dropped faster than it recovered due to asymmetric updates, routing even more gradient to the encoder.