

Dynamic Adaptive Fusion Model (DAFM): Complete Algorithm Suite for Real-Time Forecasting

Algorithm S1. Suite Overview

This document presents the complete DAFM framework consisting of:

- **Algorithm 1:** Master Coordination and Real-Time Adaptation Cycle
- **Algorithm 2:** Neural Gating Mechanism and Context Encoding
- **Algorithm 3:** Dynamic Weight Adaptation Engine
- **Algorithm 4:** Drift Detection and System Monitoring
- **Algorithm 5:** Online Learning and Parameter Updates

Algorithm 1: DAFM Master Algorithm: Real-Time Coordination & Adaptation Cycle

Data: Multi-stream data $\mathcal{D} = \{T, S^1, \dots, S^N\}$, base models $\mathcal{M} = \{\text{DT}, \text{RF}, \text{XGBoost}, \text{BiLSTM}\}$

Result: Continuous adaptive forecasts $\{\tilde{y}(t)\}_{t=1}^\infty$ with uncertainty quantification

1 **Purpose:** Master orchestration of parallel processing threads with intelligent coordination, feedback loops, and resilience mechanisms.

2 **INITIALIZATION PHASE**

3 **System Components:**

4 Initialize streaming infrastructure for multi-source data $\mathcal{D}$
 5 Load base models $\mathcal{M} = \{M_1, M_2, M_3, M_4\}$ with $M = 4$
 6 Initialize neural gating network G_θ (see Algorithm 2)
 7 Set initial adaptation fusion weights $w_i(0) = \frac{1}{M}, \forall i \in \{1, \dots, M\}$

8 **Hyperparameters:**

9 Momentum $\gamma \in [0.7, 0.9]$, window size $W \in [24, 36]$
 10 Learning rates $\eta_w = 0.01, \eta_\theta = 0.001$
 11 Context-performance balance $\alpha = 0.6$
 12 Adaptation interval $\Delta t = 50$, confidence level $p = 0.95$
 13 Drift thresholds $\delta_{\text{drift}} = 0.15, \delta_{\text{perf}} = 0.20$
 14 Regularization $\tau = 0.01$, stability $\varepsilon = 10^{-8}$

15 **Data Structures:**

16 Circular buffers: $\mathcal{B}_{\text{pred}}, \mathcal{B}_{\text{error}}, \mathcal{B}_{\text{true}}$ (size W)
 17 Concurrent queues: Q_1, Q_2, Q_3, Q_4 for inter-thread communication
 18 State dictionaries: drift_history, performance_log, weight_trajectory

19 **PARALLEL THREAD INITIALIZATION**

20 $\text{THREAD}_1 \leftarrow \text{LaunchThread}(\text{DATAINGESTIONPIPELINE})$
 21 $\text{THREAD}_2 \leftarrow \text{LaunchThread}(\text{NEURALGATINGENGINE}, \text{Algorithm 2})$
 22 $\text{THREAD}_3 \leftarrow \text{LaunchThread}(\text{WEIGHTADAPTATIONENGINE}, \text{Algorithm 3})$
 23 $\text{THREAD}_4 \leftarrow \text{LaunchThread}(\text{MONITORINGANDDRIFTDETECTION}, \text{Algorithm 4})$
 24 $\text{THREAD}_5 \leftarrow \text{LaunchThread}(\text{ONLINELEARNINGMODULE}, \text{Algorithm 5})$

25 **MAIN COORDINATION LOOP**

26 **for** each time step $t = 1, 2, \dots$ **do**

27 **Step 1: Synchronized Data Flow**

28 $\mathbf{X}(t) \leftarrow Q_1.\text{receive}() \triangleright \text{Engineered features from } \text{THREAD}_1$
 29 $\{\beta_{\text{raw}}(t), \hat{y}_i(t)\}_{i=1}^M \leftarrow Q_2.\text{receive}() \triangleright \text{From } \text{THREAD}_2$
 30 $\{w_i(t)\}_{i=1}^M \leftarrow Q_3.\text{receive}() \triangleright \text{Adapted weights from } \text{THREAD}_3$
 31 $\{\sigma(t), \text{drift_flag}(t)\} \leftarrow Q_4.\text{receive}() \triangleright \text{From } \text{THREAD}_4$

32 **Step 2: Adaptation Fusion**

33 Compute adaptation fusion forecast:

$$\tilde{y}(t) = \sum_{i=1}^M w_i(t) \hat{y}_i(t)$$

Calculate $(1 - p)$ confidence interval using $\sigma(t)$:

$$\text{CI}_{1-p}(t) = [\tilde{y}(t) - z_{p/2}\sigma(t), \tilde{y}(t) + z_{p/2}\sigma(t)]$$

34 **Step 3: Real-Time Adaptation Trigger**

35 **if** $y_{\text{true}}(t)$ becomes available **then**

36 Compute adaptation fusion error: $e(t) = y_{\text{true}}(t) - \tilde{y}(t)$
 37 Compute individual errors: $e_i(t) = y_{\text{true}}(t) - \hat{y}_i(t), \forall i$
 38 Update buffers: $\mathcal{B}_{\text{pred}}, \mathcal{B}_{\text{error}}, \mathcal{B}_{\text{true}}$
 39 TriggerOnlineLearning($\text{THREAD}_5, \{e(t), e_i(t), \mathbf{X}(t)\}$)

40 **Step 4: Intelligent Feedback Propagation**

41 Aggregate system feedback:

$$\mathcal{F}(t) = \{w(t), e(t), \{e_i(t)\}, \beta_{\text{raw}}(t), \text{drift_flag}(t)\}$$

42 $Q_2.\text{send}(\mathcal{F}(t)) \triangleright \text{Neural gate learning signal}$

43 $Q_3.\text{send}(\mathcal{F}(t)) \triangleright \text{Weight recalibration signal}$

44 $Q_4.\text{send}(\mathcal{F}(t)) \triangleright \text{Drift detector update}$

45 **Step 5: System Output & Monitoring**

46 Output($\tilde{y}(t), \text{CI}(t), \{w_i(t)\}, \text{system_health}(t)$)

47 LogMetrics(MAE(t), RMSE(t), coverage_prob(t))

48 UpdateDashboard(weight_evolution, prediction_intervals)

49 **Step 6: Resilience Check**

Algorithm 2: Neural Gating Mechanism and Context Encoding Engine

Data: Feature vector $\mathbf{X}(t) \in \mathbb{R}^d$, gating network parameters θ , base models $\mathcal{M}$
Result: Context-aware attention scores $\beta_{\text{raw}}(t) \in \mathbb{R}^M$, base predictions $\{\hat{y}_i(t)\}_{i=1}^M$

1 **Purpose:** Learn context-dependent model importance through multi-head attention mechanism.

2 **Continuous Operation Loop:**

3 **while** *system_active* **do**

4 1. **Receive Input**

5 $\mathbf{X}(t) \leftarrow Q_1.\text{receive}()$ $\triangleright$ Wait for features from `THREAD1`

6 2. **Base Model Prediction**

7 **for** $i = 1$ **to** M **do**

8 $\hat{y}_i(t) \leftarrow M_i(\mathbf{X}(t))$ $\triangleright$ Generate predictions from each model

9 3. **Multi-Head Attention Gating**

10 *Input Embedding:*

$\mathbf{h}_{\text{input}} = \text{ReLU}(\mathbf{W}_1 \mathbf{X}(t) + \mathbf{b}_1)$

11 *Multi-Head Attention: for head* $h = 1$ **to** H **do**

12 $\mathbf{Q}^{(h)} = \mathbf{W}_Q^{(h)} \mathbf{h}_{\text{input}}$

13 $\mathbf{K}^{(h)} = \mathbf{W}_K^{(h)} \mathbf{h}_{\text{input}}$

14 $\mathbf{V}^{(h)} = \mathbf{W}_V^{(h)} \mathbf{h}_{\text{input}}$

15 $\mathbf{A}^{(h)} = \text{softmax} \left(\frac{\mathbf{Q}^{(h)} (\mathbf{K}^{(h)})^\top}{\sqrt{d_k}} \right) \mathbf{V}^{(h)}$

16 *Concatenate heads:*

$\mathbf{h}_{\text{attn}} = \text{Concat}(\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(H)})$

17 *Output projection:*

$\beta_{\text{raw}}(t) = \mathbf{W}_{\text{out}} \mathbf{h}_{\text{attn}} + \mathbf{b}_{\text{out}}$

18 where $\beta_{\text{raw}}(t) \in \mathbb{R}^M$ (one score per base model)

19 4. **Broadcast Results**

20 $Q_2.\text{send}(\{\beta_{\text{raw}}(t), \hat{y}_i(t)\}_{i=1}^M)$ $\triangleright$ To `THREAD3` for weight fusion

21 5. **Receive Feedback for Learning**

22 **if** $Q_2.\text{has_feedback}()$ **then**

23 $\mathcal{F}(t) \leftarrow Q_2.\text{receive_feedback}()$

24 Extract: $e(t), \{e_i(t)\}$ from $\mathcal{F}(t)$

25 *Backpropagation:*

$L(t) = e^2(t) + \tau \|\theta\|_2^2$

$\theta \leftarrow \theta - \eta_\theta \nabla_\theta L(t)$

Algorithm 3: Dynamic Weight Adaptation Engine

Data: Context scores $\beta_{\text{raw}}(t)$, error history $\mathcal{B}_{\text{error}}$, previous weights $\{w_i(t-1)\}$

Result: Adapted fusion weights $\{w_i(t)\}_{i=1}^M$

1 **Purpose:** Fuse context-awareness with historical performance using momentum stabilization.

2 **Continuous Operation Loop:**

3 **while** *system_active* **do**

4 1. **Receive Inputs**

5 $\{\beta_{\text{raw}}(t), \hat{y}_i(t)\}_{i=1}^M \leftarrow Q_2.\text{receive}()$

6 2. **Rolling Performance Evaluation**

7 **for** $i = 1$ **to** M **do**

8 Retrieve error history from buffer $\mathcal{B}_{\text{error}}$

9 Compute rolling RMSE over window W :

$$\text{RMSE}_i(t) = \sqrt{\frac{1}{W} \sum_{s=t-W}^{t-1} (y_{\text{true}}(s) - \hat{y}_i(s))^2}$$

10 Normalize to performance scores:

$$P_i(t) = \frac{1/\text{RMSE}_i(t)}{\sum_{k=1}^M 1/\text{RMSE}_k(t)}$$

11 3. **Context-Performance Hybrid Fusion**

12 **for** $i = 1$ **to** M **do**

13 Combine context and performance in log-space:

$$\tilde{\beta}_i(t) = \alpha \beta_{\text{raw},i}(t) + (1 - \alpha) \log(P_i(t) + \varepsilon)$$

14 Apply softmax normalization:

$$\beta_i(t) = \frac{\exp(\tilde{\beta}_i(t))}{\sum_{j=1}^M \exp(\tilde{\beta}_j(t))}$$

15 4. **Momentum-Based Stabilization**

16 **for** $i = 1$ **to** M **do**

17 Apply exponential moving average:

$$w_i(t) = \gamma w_i(t-1) + (1 - \gamma) \beta_i(t)$$

18 Enforce non-negativity and normalization:

$$w_i(t) = \frac{\max(w_i(t), 0)}{\sum_{j=1}^M \max(w_j(t), 0)}$$

19 5. **Regularization Check**

20 **if** $\max_i w_i(t) > 0.85$ **then**

21 Apply entropy regularization to prevent over-concentration:

$$w_i(t) \leftarrow w_i(t) \cdot (1 - \lambda) + \frac{\lambda}{M}$$

 where $\lambda = 0.1$ (regularization strength)

22 Renormalize weights

23 6. **Broadcast Weights**

24 $Q_3.\text{send}(\{w_i(t)\}_{i=1}^M) \triangleright$ To master coordination loop

25 7. **Receive Feedback**

26 **if** $Q_3.\text{has_feedback}()$ **then**

27 $\mathcal{F}(t) \leftarrow Q_3.\text{receive_feedback}()$

28 Update weight trajectory history

Algorithm 4: Drift Detection and System Monitoring

Data: Adaptation fusion forecast $\tilde{y}(t)$, base predictions $\{\hat{y}_i(t)\}$, weights $\{w_i(t)\}$

Result: Uncertainty estimate $\sigma(t)$, drift flags, system health status

1 **Purpose:** Monitor prediction uncertainty, detect concept drift, and ensure system resilience.

2 **Continuous Monitoring Loop:**

3 **while** *system_active* **do**

4 1. **Receive Predictions**

5 $\tilde{y}(t) \leftarrow \text{fusion_forecast}$

6 $\{\hat{y}_i(t), w_i(t)\}_{i=1}^M \leftarrow Q_3.\text{receive}()$

7 2. **Uncertainty Quantification**

8 Compute adaptation fusion variance:

$$\sigma^2(t) = \sum_{i=1}^M w_i(t) (\hat{y}_i(t) - \tilde{y}(t))^2$$

 Compute prediction disagreement:

$$D(t) = \max_{i,j} |\hat{y}_i(t) - \hat{y}_j(t)|$$

9 3. **Concept Drift Detection (ADWIN Test)**

10 Maintain sliding window of recent errors: $\mathcal{W}_{\text{error}}$

11 Split window into two subwindows: $\mathcal{W}_1, \mathcal{W}_2$

12 Compute means: $\mu_1 = \text{mean}(\mathcal{W}_1)$, $\mu_2 = \text{mean}(\mathcal{W}_2)$

13 Compute drift statistic:

$$\text{drift_score}(t) = \frac{|\mu_1 - \mu_2|}{\sqrt{\text{var}(\mathcal{W}_{\text{error}})}}$$

if $\text{drift_score}(t) > \delta_{\text{drift}}$ **then**

14 | $\text{drift_flag}(t) \leftarrow \text{TRUE}$

15 | $\text{LogDriftEvent}(t, \text{drift_score}(t))$

16 | $\text{TriggerAdaptiveRecalibration}()$

17 4. **Performance Degradation Check**

18 Compute recent MAE over last 10 steps:

$$\text{MAE}_{\text{recent}} = \frac{1}{10} \sum_{s=t-10}^{t-1} |y_{\text{true}}(s) - \tilde{y}(s)|$$

if $\text{MAE}_{\text{recent}} / \text{MAE}_{\text{baseline}} > 1 + \delta_{\text{perf}}$ **then**

19 | $\text{perf_degradation_flag}(t) \leftarrow \text{TRUE}$

20 | $\text{TriggerModelRetraining}()$

21 5. **System Health Assessment**

22 $\text{health_score} \leftarrow \text{ComputeHealthScore}(\sigma(t), \text{drift_flag}, \text{perf_flag})$

23 **if** $\text{health_score} < 0.3$ **then**

24 | $\text{system_health}(t) \leftarrow \text{CRITICAL}$

25 **else if** $\text{health_score} < 0.6$ **then**

26 | $\text{system_health}(t) \leftarrow \text{WARNING}$

27 **else**

28 | $\text{system_health}(t) \leftarrow \text{HEALTHY}$

29 6. **Broadcast Monitoring Results**

30 $Q_4.\text{send}(\{\sigma(t), \text{drift_flag}(t), \text{system_health}(t)\})$

Algorithm 5: Online Learning and Parameter Update Module

Data: Fusion error $e(t)$, individual errors $\{e_i(t)\}$, current weights $\{w_i(t)\}$

Result: Updated weights and gating parameters for continuous improvement

```

1 Purpose: Perform gradient-based online learning for immediate error correction.
2 Triggered Learning Loop:
3   while system_active do
4     1. Wait for Learning Trigger
5       WaitForTrigger() ▷ Triggered when ground truth arrives
6       Receive:  $\{e(t), e_i(t), \mathbf{X}(t), y_{\text{true}}(t)\}$ 
7     2. Weight Gradient Computation
8       for  $i = 1$  to  $M$  do
9         Compute weight sensitivity to error:
10
11           
$$\Delta w_i(t) = \eta_w \cdot e(t) \cdot (y_{\text{true}}(t) - \hat{y}_i(t))$$

12
13           Intuition: Increase weight for models whose predictions
14           would have reduced fusion error
15     3. Immediate Weight Update
16       for  $i = 1$  to  $M$  do
17          $w_i(t) \leftarrow w_i(t) + \Delta w_i(t)$ 
18
19         Clip to valid range:  $w_i(t) \leftarrow \max(0, \min(1, w_i(t)))$ 
20         Renormalize:
21
22           
$$w_i(t) \leftarrow \frac{w_i(t)}{\sum_{j=1}^M w_j(t)}$$

23
24     4. Neural Gate Fine-Tuning
25       Compute gating loss:
26
27           
$$L_{\text{gate}}(t) = e^2(t) + \tau \|\theta\|_2^2$$

28
29       Compute gradient via backpropagation:
30
31           
$$\nabla_{\theta} L_{\text{gate}}(t) = \frac{\partial L_{\text{gate}}}{\partial \theta}$$

32
33       Update gating parameters:
34
35           
$$\theta \leftarrow \theta - \eta_{\theta} \nabla_{\theta} L_{\text{gate}}(t)$$

36
37     5. Individual Model Performance Tracking
38       for  $i = 1$  to  $M$  do
39         Update individual model error history
40         Compute exponential moving average of errors:
41
42           
$$\bar{e}_i(t) = 0.9 \cdot \bar{e}_i(t-1) + 0.1 \cdot |e_i(t)|$$

43
44     6. Adaptive Learning Rate Scheduling
45       if  $t \bmod 100 = 0$  then
46         Analyze convergence behavior over last 100 steps
47         if oscillating_pattern_detected then
48            $\eta_w \leftarrow 0.8 \cdot \eta_w$ 
49            $\eta_{\theta} \leftarrow 0.8 \cdot \eta_{\theta}$ 
50
51         if slow_convergence_detected then
52            $\eta_w \leftarrow 1.1 \cdot \eta_w$  (max: 0.1)
53            $\eta_{\theta} \leftarrow 1.1 \cdot \eta_{\theta}$  (max: 0.01)
54
55     7. Propagate Updates
56       Broadcast updated weights to THREAD3
57       Broadcast updated  $\theta$  to THREAD2
58       Update global state dictionary

```

Theoretical Guarantees

Convergence: Under mild regularity conditions (bounded gradients, Lipschitz continuity of loss), the online learning module ensures:

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T e^2(t) \leq \epsilon_{\text{opt}} + O(\eta_w)$$

Stability: The momentum term γ prevents weight oscillations, guaranteeing:

$$\|w(t) - w(t-1)\|_2 \leq (1 - \gamma) \|\beta(t) - \beta(t-1)\|_2$$

Drift Adaptation: The ADWIN drift detector provides probabilistic guarantees on false alarm rate $< \delta$ while detecting drift with delay $O(\log(1/\delta))$.