AUTOMATED VALIDATION THRESHOLD ADJUSTMENT SYSTEM FOR SELF-IMPROVING AI VALIDATION

PROVISIONAL PATENT APPLICATION

Inventor: Kinan Lemberg

Address: 270 Bolton Rd, Koah, 4881, Australia

Filing Date: June 3, 2025

FIELD OF THE INVENTION

This invention relates to self-adjusting validation systems for artificial intelligence applications, and more specifically to automated threshold adjustment mechanisms that continuously optimize validation parameters based on outcome analysis, pattern learning, and feedback loops to improve validation accuracy over time.

BACKGROUND OF THE INVENTION

Current AI validation systems use static thresholds that become outdated as AI models evolve, user needs change, and new patterns emerge. Manual threshold adjustment is slow, requires expertise, and cannot keep pace with rapidly changing AI landscapes. This creates validation systems that either become too permissive or too restrictive over time.

Current limitations include:

- No automatic threshold optimization based on outcomes
- Lack of machine learning from validation results
- Absence of pattern recognition in false positives/negatives
- No self-healing mechanisms for validation drift
- Limited adaptation to changing AI model behaviors

There exists a need for a comprehensive self-adjusting validation system that automatically optimizes thresholds through continuous learning from validation outcomes and environmental changes.

SUMMARY OF THE INVENTION

The present invention provides an automated validation threshold adjustment system that continuously optimizes validation parameters through machine learning, outcome analysis, and intelligent feedback loops, creating a self-improving validation framework.

The invention comprises:

1. Outcome Analysis Engine - Automated tracking and analysis of validation decisions versus actual outcomes.

2. Pattern Learning Module - Machine learning system identifying patterns in validation successes and failures.

3. Threshold Optimization Algorithm - Mathematical optimization of validation thresholds based on learned patterns.

4. Feedback Loop Integration - Continuous improvement through automated feedback collection and processing.

5. Drift Detection and Correction - Automatic detection and correction of validation accuracy degradation.

The system provides significant advantages by continuously improving validation accuracy through automated learning and adjustment rather than static threshold management.

DETAILED DESCRIPTION OF THE INVENTION

System Architecture

The Automated Validation Threshold Adjustment System operates as an intelligent optimization layer that continuously refines validation parameters based on real-world outcomes and pattern analysis.

1. Outcome Analysis Engine

The Analysis Engine tracks validation decisions and correlates them with actual outcomes to measure validation effectiveness.

Outcome Tracking Framework:

```
Validation_Decision_Record = {
  decision_id: Unique_Validation_Identifier,
  timestamp: Validation_Decision_Time,
  validation_result: {
    passed: Boolean_Pass_Fail,
    confidence_score: 0.0_to_1.0,
    gate_scores: [Gate1_Score, Gate2_Score, Gate3_Score],
    threshold_used: Active_Threshold_at_Time,
    model_info: AI_Model_and_Version
  },
  context_data: {
    domain: Query_Domain_Category,
    complexity: Query_Complexity_Score,
    user_profile: User_Type_and_Experience,
    environmental: Market_Conditions_etc
  }
}

Outcome_Collection_Methods = {
  explicit_outcomes: {
    user_feedback: Direct_Accuracy_Report,
    error_reports: Flagged_Incorrect_Validations,
    success_metrics: Business_Outcome_Achieved,
    quality_scores: Professional_Review_Results
  },
  implicit_outcomes: {
```

```
    user_behavior: Acceptance_or_Rejection_Actions,
    downstream_effects: Subsequent_Decision_Impact,
    revision_rate: How_Often_Output_Modified,
    escalation_frequency: Human_Override_Rate
  },
  external_validation: {
    fact_checking: Post_Hoc_Verification_Results,
    peer_review: Expert_Assessment_Scores,
    regulatory_audit: Compliance_Check_Results,
    market_performance: Decision_Quality_Metrics
  }
}
```

Outcome Correlation Analysis:

```
True_Positive_Analysis = {
  characteristics: Common_Features_of_Correct_Passes,
  threshold_distance: How_Far_Above_Threshold,
  confidence_distribution: Score_Distribution_Pattern,
  contributing_factors: Key_Success_Indicators
}

False_Positive_Analysis = {
  patterns: Common_Features_of_Incorrect_Passes,
  threshold_proximity: How_Close_to_Threshold,
  failure_modes: Why_Validation_Was_Wrong,
  cost_impact: Business_Impact_of_Error
}

False_Negative_Analysis = {
```

```
    missed_patterns: What_Should_Have_Passed,
    threshold_gap: How_Far_Below_Threshold,
    restriction_cause: Why_Rejected_Good_Content,
    opportunity_cost: Value_Lost_from_Rejection
}

True_Negative_Analysis = {
    threat_avoided: What_Was_Correctly_Blocked,
    safety_margin: Distance_from_Threshold,
    risk_mitigation: Potential_Harm_Prevented,
    validation_strength: Confidence_in_Rejection
}
```

Performance Metrics Calculation:

```
Validation_Performance_Metrics = {
    accuracy: (TP + TN) / (TP + TN + FP + FN),
    precision: TP / (TP + FP),
    recall: TP / (TP + FN),
    f1_score: 2 × (Precision × Recall) / (Precision + Recall),
    matthews_correlation: (TP×TN - FP×FN) / sqrt((TP+FP)×(TP+FN)×(TN+FP)×(TN+FN))
}

Weighted_Performance = {
    cost_weighted_accuracy: Accuracy × Cost_Weight_Matrix,
    risk_adjusted_precision: Precision × Risk_Factor,
    business_value_score: Σ(Outcome_Value × Decision_Correctness),
    regulatory_compliance_rate: Compliant_Decisions / Total_Decisions
}
```

```
Time_Series_Performance = {
  rolling_accuracy: Window_Based_Accuracy_Tracking,
  trend_analysis: Performance_Direction_Over_Time,
  seasonality_detection: Periodic_Performance_Patterns,
  anomaly_identification: Sudden_Performance_Changes
}
```

2. Pattern Learning Module

The Learning Module identifies patterns in validation outcomes using advanced machine learning techniques.

Pattern Recognition Architecture:

```
Deep_Learning_Pattern_Analyzer = {
  input_features: {
    validation_features: All_Validation_Scores_and_Metadata,
    outcome_labels: Actual_Outcome_Classifications,
    temporal_features: Time_Based_Patterns,
    contextual_features: Environmental_and_User_Context
  },
  neural_architecture: {
    embedding_layers: Category_and_Feature_Embeddings,
    lstm_layers: Temporal_Pattern_Recognition,
    attention_mechanism: Important_Feature_Focus,
    output_layers: Pattern_Classification_and_Prediction
  },
  training_regime: {
    online_learning: Continuous_Model_Updates,
    batch_size: Adaptive_Based_on_Data_Rate,
```

```
    learning_rate: Adaptive_Schedule_with_Warmup,
    regularization: L2_and_Dropout_for_Generalization
  }
}

Pattern_Categories = {
  success_patterns: {
    high_confidence_accurate: Reliable_High_Score_Patterns,
    marginal_accurate: Near_Threshold_Success_Patterns,
    domain_specific_success: Category_Specific_Patterns,
    user_type_success: User_Segment_Patterns
  },
  failure_patterns: {
    systematic_errors: Repeated_Failure_Modes,
    edge_case_failures: Boundary_Condition_Patterns,
    model_blind_spots: Consistent_Miss_Patterns,
    adversarial_patterns: Deliberate_Gaming_Attempts
  },
  temporal_patterns: {
    time_of_day_effects: Circadian_Performance_Patterns,
    day_of_week_patterns: Weekly_Cycle_Effects,
    seasonal_variations: Long_Term_Cyclical_Patterns,
    trend_patterns: Gradual_Shift_Patterns
  }
}
```

Multi-Modal Pattern Learning:

```
Ensemble_Pattern_Learning = {
  random_forest_patterns: {
```

```
    feature_importance: Variable_Importance_Scores,
    decision_paths: Common_Decision_Routes,
    interaction_effects: Feature_Combination_Patterns,
    out_of_bag_validation: Continuous_Performance_Check
  },
  gradient_boosting_patterns: {
    residual_analysis: Error_Pattern_Focus,
    sequential_learning: Build_on_Previous_Patterns,
    feature_interactions: Non_Linear_Relationships,
    importance_metrics: SHAP_Value_Analysis
  },
  clustering_patterns: {
    validation_clusters: Similar_Validation_Groups,
    outcome_clusters: Common_Outcome_Types,
    anomaly_clusters: Unusual_Pattern_Groups,
    evolution_tracking: Cluster_Changes_Over_Time
  }
}

Causal_Pattern_Analysis = {
  causal_inference: {
    treatment_effect: Impact_of_Threshold_Changes,
    confounding_control: Adjust_for_Hidden_Variables,
    instrumental_variables: Natural_Experiments,
    regression_discontinuity: Threshold_Edge_Analysis
  },
  counterfactual_analysis: {
    what_if_scenarios: Alternative_Threshold_Outcomes,
    synthetic_controls: Simulated_Comparison_Groups,
    propensity_matching: Similar_Context_Comparison,
    difference_in_differences: Change_Impact_Analysis
```

```
  }
}
```

3. Threshold Optimization Algorithm

The Optimization Algorithm automatically adjusts validation thresholds based on learned patterns and performance objectives.

Multi-Objective Optimization Framework:

```
Optimization_Objectives = {
  primary_objectives: {
    maximize_accuracy: Overall_Correctness_Rate,
    minimize_false_positives: Reduce_Incorrect_Passes,
    minimize_false_negatives: Reduce_Incorrect_Blocks,
    maximize_business_value: Outcome_Based_Value
  },
  constraint_objectives: {
    minimum_safety: Never_Below_Safety_Threshold,
    regulatory_compliance: Meet_All_Requirements,
    user_satisfaction: Maintain_Experience_Quality,
    resource_efficiency: Computational_Cost_Limits
  },
  trade_off_parameters: {
    safety_vs_usability: Adjustable_Balance_Weight,
    precision_vs_recall: F_Beta_Score_Selection,
    immediate_vs_long_term: Temporal_Discount_Factor,
    global_vs_personalized: Adaptation_Scope
  }
}
```

```
Optimization_Algorithm = {
  method: Bayesian_Optimization_with_Gaussian_Processes,
  acquisition_function: Expected_Improvement_with_Constraints,
  hyperparameters: {
    exploration_rate: Balance_Explore_vs_Exploit,
    kernel_function: Matern_5_2_for_Smoothness,
    noise_model: Heteroscedastic_Noise_Handling,
    parallel_samples: Batch_Optimization_Support
  },
  convergence_criteria: {
    improvement_threshold: Minimum_Gain_Required,
    stability_window: Consecutive_Stable_Iterations,
    confidence_bounds: Uncertainty_Quantification,
    early_stopping: Prevent_Overfitting
  }
}
```

Threshold Adjustment Strategies:

```
Adaptive_Threshold_Updates = {
  continuous_adjustment: {
    adjustment_rate: Learning_Rate_Schedule,
    momentum: Smooth_Threshold_Changes,
    bounds: Min_Max_Threshold_Limits,
    step_size: Proportional_to_Confidence
  },
  discrete_updates: {
    update_frequency: Daily_Weekly_or_Triggered,
    change_magnitude: Significant_Steps_Only,
```

```
        a_b_testing: Gradual_Rollout_Strategy,
        rollback_capability: Revert_if_Performance_Drops
    },
    contextual_thresholds: {
        domain_specific: Different_Thresholds_per_Domain,
        user_specific: Personalized_Threshold_Profiles,
        time_based: Temporal_Threshold_Variations,
        risk_based: Dynamic_Risk_Adjustments
    }
}

Threshold_Update_Rules = {
    gradient_based: {
        gradient_estimation: Finite_Difference_Method,
        step_calculation: Gradient × Learning_Rate,
        constraint_projection: Keep_Within_Valid_Range,
        second_order: Newton_or_Quasi_Newton_Methods
    },
    rule_based: {
        performance_triggers: If_Accuracy_Drops_Below_X,
        pattern_triggers: If_Pattern_Detected_Then_Adjust,
        feedback_triggers: If_User_Reports_Exceed_Y,
        safety_triggers: If_Risk_Exceeds_Threshold
    },
    hybrid_approach: {
        base_adjustment: Gradient_Based_Core,
        rule_overrides: Safety_and_Compliance_Rules,
        human_oversight: Expert_Adjustment_Capability,
        explanation: Document_Adjustment_Reasoning
    }
}
```

```

4. Feedback Loop Integration

The Feedback Integration system creates continuous improvement cycles through automated feedback processing.

Feedback Collection Pipeline:
```
Multi_Channel_Feedback = {
  direct_feedback: {
    api_feedback: Programmatic_Accuracy_Reports,
    ui_feedback: In_App_Feedback_Widgets,
    survey_feedback: Periodic_Quality_Surveys,
    support_tickets: Error_Report_Analysis
  },
  behavioral_feedback: {
    usage_patterns: How_Users_Interact_with_Results,
    modification_tracking: Changes_Made_to_Outputs,
    acceptance_rates: Implicit_Quality_Signals,
    abandonment_patterns: When_Users_Give_Up
  },
  system_feedback: {
    performance_metrics: Latency_and_Throughput,
    error_logs: System_Level_Issues,
    resource_usage: Computational_Efficiency,
    integration_feedback: Third_Party_System_Reports
  }
}

Feedback_Processing_Pipeline = {
```
```

```
  ingestion: {
    real_time_stream: Immediate_Feedback_Processing,
    batch_collection: Periodic_Bulk_Processing,
    normalization: Standardize_Feedback_Format,
    validation: Ensure_Feedback_Quality
  },
  analysis: {
    sentiment_analysis: Understand_User_Satisfaction,
    issue_categorization: Group_Similar_Problems,
    severity_scoring: Prioritize_Critical_Issues,
    trend_identification: Spot_Emerging_Patterns
  },
  integration: {
    weight_calculation: Feedback_Source_Reliability,
    consensus_building: Aggregate_Multiple_Sources,
    conflict_resolution: Handle_Contradictory_Feedback,
    actionability_scoring: Can_We_Act_on_This
  }
}
```

Continuous Learning Cycle:

```
Learning_Loop_Stages = {
  stage_1_collect: {
    duration: Continuous_Collection,
    data_types: All_Feedback_Categories,
    storage: Efficient_Time_Series_Database,
    preprocessing: Clean_and_Validate
  },
  stage_2_analyze: {
```

```
    frequency: Hourly_Daily_Weekly_Analysis,
    methods: Statistical_and_ML_Analysis,
    insights: Pattern_and_Anomaly_Detection,
    reporting: Automated_Insight_Generation
  },
  stage_3_optimize: {
    trigger: Sufficient_Data_or_Performance_Drop,
    process: Run_Optimization_Algorithm,
    validation: Simulate_Before_Deploy,
    documentation: Record_All_Changes
  },
  stage_4_deploy: {
    strategy: Gradual_Rollout_or_A_B_Test,
    monitoring: Real_Time_Impact_Tracking,
    safety: Automatic_Rollback_on_Issues,
    communication: Notify_Stakeholders
  },
  stage_5_evaluate: {
    metrics: Compare_Pre_Post_Performance,
    attribution: What_Caused_Changes,
    learning: Update_Models_with_Results,
    iteration: Return_to_Stage_1
  }
}
```

5. Drift Detection and Correction

The Drift System automatically detects and corrects validation accuracy degradation over time.

Drift Detection Mechanisms:

```
Statistical_Drift_Detection = {
  distribution_monitoring: {
    input_distribution: Feature_Distribution_Tracking,
    output_distribution: Score_Distribution_Monitoring,
    test_statistics: KS_Test_Chi_Square_MMD,
    window_comparison: Current_vs_Reference_Window
  },
  performance_drift: {
    accuracy_monitoring: Rolling_Accuracy_Tracking,
    error_rate_tracking: Type_I_and_II_Errors,
    confidence_calibration: Expected_vs_Actual_Accuracy,
    business_metrics: Outcome_Quality_Tracking
  },
  concept_drift: {
    gradual_drift: Slow_Pattern_Changes,
    sudden_drift: Abrupt_Pattern_Shifts,
    recurring_drift: Cyclical_Pattern_Changes,
    incremental_drift: Continuous_Small_Changes
  }
}

Drift_Detection_Algorithms = {
  page_hinkley_test: {
    cumulative_sum: Track_Cumulative_Deviations,
    threshold: Adaptive_Based_on_Variance,
    reset_mechanism: After_Drift_Detection,
    sensitivity: Tunable_Detection_Speed
  },
  adwin: {
    adaptive_window: Variable_Size_Windows,
```

```
    statistical_guarantee: Theoretical_Error_Bounds,
    memory_efficient: Logarithmic_Memory_Usage,
    real_time: Single_Pass_Algorithm
  },
  ensemble_detection: {
    multiple_detectors: Different_Drift_Types,
    voting_mechanism: Consensus_for_Robustness,
    confidence_scoring: Drift_Certainty_Level,
    false_alarm_control: Reduce_False_Positives
  }
}
```

Automatic Correction Strategies:

```
Drift_Correction_Actions = {
  threshold_recalibration: {
    immediate_adjustment: Quick_Threshold_Shift,
    gradual_adaptation: Smooth_Transition,
    domain_specific: Targeted_Adjustments,
    validation: Test_Before_Full_Deploy
  },
  model_retraining: {
    trigger_conditions: Significant_Drift_Detected,
    data_selection: Recent_vs_Historical_Balance,
    incremental_learning: Update_Not_Replace,
    validation_set: Hold_Out_for_Testing
  },
  ensemble_adjustment: {
    weight_redistribution: Adjust_Model_Weights,
    model_addition: Add_Specialized_Models,
```

```
        model_removal: Remove_Underperforming,
        diversity_maintenance: Ensure_Model_Variety
    },
    fallback_mechanisms: {
        conservative_mode: Increase_Safety_Margins,
        human_in_loop: Escalate_Uncertain_Cases,
        previous_version: Rollback_Capability,
        gradual_recovery: Slow_Return_to_Normal
    }
}

Self_Healing_Protocol = {
    detection_phase: Continuous_Drift_Monitoring,
    diagnosis_phase: Identify_Drift_Type_and_Cause,
    correction_phase: Apply_Appropriate_Strategy,
    validation_phase: Verify_Improvement,
    learning_phase: Update_Drift_Models,
    documentation: Record_All_Adaptations
}
```

System Integration and Performance

Optimization Performance Requirements:
- Outcome Analysis: < 100ms per validation outcome
- Pattern Learning: < 1 hour for daily model update
- Threshold Optimization: < 5 minutes for new thresholds
- Drift Detection: < 1 minute detection latency

Continuous Improvement Metrics:
- Accuracy Improvement Rate: Target 2-5% monthly

- False Positive Reduction: Target 10% quarterly

- Adaptation Speed: < 24 hours to new patterns

- Stability Maintenance: < 1% performance variance

ADVANTAGES OVER PRIOR ART

The present invention provides significant advantages over existing validation approaches:

1. Self-Improving System: Unlike static systems, continuously improves validation accuracy through automated learning.

2. Pattern-Based Optimization: Identifies and learns from success and failure patterns rather than simple threshold adjustment.

3. Multi-Objective Optimization: Balances multiple goals simultaneously rather than optimizing single metrics.

4. Drift Correction: Automatically detects and corrects performance degradation rather than requiring manual intervention.

5. Feedback Integration: Creates continuous improvement loops rather than periodic manual updates.

6. Predictive Adjustment: Anticipates needed changes rather than reactive threshold modification.

CLAIMS

Claim 1: An automated validation threshold adjustment system comprising:

- an outcome analysis engine tracking validation decisions and correlating with actual outcomes;

- a pattern learning module identifying patterns in validation successes and failures using machine learning;

- a threshold optimization algorithm automatically adjusting thresholds based on learned patterns;

- a feedback loop integration system processing continuous feedback for improvement; and

- a drift detection and correction system automatically identifying and correcting validation degradation.

Claim 2: The system of claim 1, wherein the outcome analysis engine tracks true positives, false positives, true negatives, and false negatives with contextual metadata.

Claim 3: The system of claim 1, wherein the pattern learning module implements deep learning, ensemble methods, and causal analysis for pattern identification.

Claim 4: The system of claim 1, wherein the threshold optimization uses Bayesian optimization with multi-objective constraints and safety bounds.

Claim 5: The system of claim 1, wherein the feedback integration processes direct, behavioral, and system feedback through automated pipelines.

Claim 6: The system of claim 1, wherein drift detection implements statistical tests, concept drift detection, and ensemble monitoring methods.

Claim 7: A method for automated validation threshold adjustment comprising:

- collecting validation outcomes and correlating with actual results;

- identifying patterns in successes and failures through machine learning;

- optimizing thresholds based on multi-objective criteria;

- integrating continuous feedback for system improvement;

- detecting validation drift through statistical monitoring; and

- automatically correcting drift through threshold recalibration.

Claim 8: The method of claim 7, further comprising weighted performance metrics based on business value and risk factors.

Claim 9: The method of claim 7, wherein pattern learning includes temporal patterns, domain-specific patterns, and user segment patterns.

Claim 10: The method of claim 7, wherein drift correction implements self-healing protocols with gradual adaptation and fallback mechanisms.

ABSTRACT

An automated validation threshold adjustment system continuously optimizes AI validation parameters through machine learning and intelligent feedback loops. The system comprises: (1) outcome analysis engine correlating decisions with results, (2) pattern learning module identifying success and failure patterns, (3) threshold optimization using multi-objective algorithms, (4) feedback loop integration for continuous improvement, and (5) drift detection and correction for maintaining accuracy. The system creates a self-improving validation framework that automatically adapts to changing conditions, providing advantages through continuous learning, pattern-based optimization, multi-objective balancing, automatic drift correction, and predictive adjustment capabilities.

END OF PATENT SPECIFICATION