Self-Evolving Small Language Model (SE-SLM)

1. Introduction

1.1 Purpose

This document describes the detailed software requirements for the Self-Evolving Small Language Model (SE-SLM), an adaptive AI system that monitors its runtime usage and autonomously optimizes its architecture by pruning, rewiring, and compressing itself without full retraining.

1.2 Scope

The system is designed to:

- Monitor inference behavior in real time
- Detect domain drift and usage patterns
- Self-optimize architecture dynamically
- Reduce compute and memory footprint
- Maintain or improve model accuracy

1.3 Definitions

SLM – Small Language Model

Evolution Cycle – Periodic optimization process

Domain Drift – Change in task/data distribution

2. Overall Description

2.1 Product Perspective

SE-SLM extends a transformer model with telemetry monitoring, structural analysis, and an evolution engine that enables self-optimization.

2.2 High-Level Components

- Base Transformer SLM
- Telemetry Monitor

• Usage Profiler

• Structural Analyzer

• Evolution Engine

• Model Recompiler

• Validation Sandbox

• Drift Detector

• Model Registry

• Governance Manager

3. Functional Requirements

3.1 Runtime Telemetry Monitoring

The system shall:

• Log token activations

• Track attention head utilization

• Measure layer execution frequency

• Capture latency and memory metrics

3.2 Usage Profiling

The system shall aggregate telemetry into:

• Frequent token paths

• Dormant neurons

• Redundant heads

• Rare vocabulary branches

3.3 Structural Analysis

The system shall:

• Identify prunable attention heads

• Detect redundant feedforward layers

• Score neuron importance

• Recommend rewiring opportunities

3.4 Evolution Engine

The engine shall:

• Execute scheduled evolution cycles

• Generate compressed architectures

• Simulate pruning impact

• Produce candidate model graphs

Supported optimizations:

• Head pruning

• Layer pruning

• Embedding compression

• Sparse rewiring

• Quantization

3.5 Self-Recompilation

The system shall recompile models using:

• Weight inheritance

• Graph rewriting

• Lightweight distillation

Outputs:

• Optimized model binary

• Architecture diff report

3.6 Validation Sandbox

Validation must include:

- Accuracy regression tests
- Domain benchmarks
- Hallucination checks
- Latency measurement

3.7 Domain Drift Detection

Drift shall be detected using:

- Embedding distribution shifts
- Vocabulary change rate
- Intent variance

3.8 Model Registry

The system shall maintain:

- Model lineage
- Compression ratios
- Accuracy deltas
- Evolution metadata

3.9 Rollback Governance

The system shall support:

- Instant rollback
- Evolution audit logs
- Change approvals

4. Non-Functional Requirements

Performance:

• ≥20% latency improvement

• ≥30% memory reduction

Reliability:

• Zero downtime deployment

• Blue-green releases

Security:

• Model encryption

• Secure telemetry storage

5. Technical Requirements

AI/ML Stack:

• PyTorch

• Hugging Face Transformers

• DeepSpeed

Telemetry:

• OpenTelemetry

• Prometheus

Backend:

• FastAPI / Node.js

Storage:

• PostgreSQL

• Object storage (S3/MinIO)

6. System Workflows

Evolution Cycle:

1. Collect telemetry

2. Aggregate metrics

3. Score components

4. Generate pruning plan

5. Recompile model

6. Validate

7. Deploy

Drift-Triggered Evolution:

1. Detect distribution shift

2. Trigger micro-evolution

3. Optimize domain paths

7. Constraints

• Safety layers cannot be pruned

• Tokenizer compatibility must remain

• Max 40% pruning per cycle

8. Risks & Mitigations

Accuracy loss → Validation sandbox

Over-pruning → Threshold caps

Model collapse → Rollback registry