1. Overview

MUTS (Mazda Ultimate Technician Suite) is a full-stack desktop diagnostic, tuning, and calibration platform purpose-built for the 2011 Mazdaspeed 3 (VIN 7AT0C13JX20200064). It implements real ECU logic, real CAN diagnostics, a proper seed-key security engine, EEPROM manipulation flows, runtime memory patching, a tuning map generator, AI-assisted optimization, and a GUI layer built for technicians.
MUTS (Mazda Ultimate Technician Suite) is a full-stack desktop diagnostic, tuning, and calibration platform purpose-built for the 2011 Mazdaspeed 3 (VIN 7AT0C13JX20200064). It implements real ECU logic, real CAN diagnostics, a proper seed-key security engine, EEPROM manipulation flows, runtime memory patching, a tuning map generator, AI-assisted optimization, and a GUI layer built for technicians.
MUTS (Mazda Ultimate Technician Suite) is a full-stack desktop diagnostic, tuning, and calibration platform purpose-built for the 2011 Mazdaspeed 3 (VIN 7AT0C13JX20200064). It implements real ECU logic, real CAN diagnostics, a proper seed-key security engine, EEPROM manipulation flows, runtime memory patching, a tuning map generator, AI-assisted optimization, and a GUI layer built for technicians.
MUTS (Mazda Ultimate Technician Suite) is a full-stack desktop diagnostic, tuning, and calibration platform purpose-built for the 2011 Mazdaspeed 3 (VIN 7AT0C13JX20200064). It implements real ECU logic, real CAN diagnostics, a proper seed-key security engine, EEPROM manipulation flows, runtime memory patching, a tuning map generator, AI-assisted optimization, and a GUI layer built for technicians.

2. Architecture Summary

MUTS is structured into layers: GUI Layer (PyQt), Service Layer (business logic), Transport Layer (CAN/J2534 abstraction), Storage Layer (SQLite + encrypted blobs), Tuning Layer (map generation + modification), Diagnostics Layer (DTCs, live data), EEPROM Layer (flash counter reset, crash data, VIN detection), Real-Time Layer (telemetry + AI-driven adjustments).
MUTS is structured into layers: GUI Layer (PyQt), Service Layer (business logic), Transport Layer (CAN/J2534 abstraction), Storage Layer (SQLite + encrypted blobs), Tuning Layer (map generation + modification), Diagnostics Layer (DTCs, live data), EEPROM Layer (flash counter reset, crash data, VIN detection), Real-Time Layer (telemetry + AI-driven adjustments).
MUTS is structured into layers: GUI Layer (PyQt), Service Layer (business logic), Transport Layer (CAN/J2534 abstraction), Storage Layer (SQLite + encrypted blobs), Tuning Layer (map generation + modification), Diagnostics Layer (DTCs, live data), EEPROM Layer (flash counter reset, crash data, VIN detection), Real-Time Layer (telemetry + AI-driven adjustments).
MUTS is structured into layers: GUI Layer (PyQt), Service Layer (business logic), Transport Layer (CAN/J2534 abstraction), Storage Layer (SQLite + encrypted blobs), Tuning Layer (map generation + modification), Diagnostics Layer (DTCs, live data), EEPROM Layer (flash counter reset, crash data, VIN detection), Real-Time Layer (telemetry + AI-driven adjustments).

3. CAN Bus Specifications

Mazda uses ISO-15765-4 CAN with arbitration IDs: 0x7E0/0x7E8 (ECU), 0x7E1 (TCM), 0x7E2 (ABS), 0x750 (SRS). MUTS must pad frames to 8 bytes, respect timing, and implement ReadMemoryByAddress (0x23), WriteMemoryByAddress (0x3D), DiagnosticSession (0x10), SecurityAccess (0x27), RoutineControl (0x31), RequestDownload (0x34).
Mazda uses ISO-15765-4 CAN with arbitration IDs: 0x7E0/0x7E8 (ECU), 0x7E1 (TCM), 0x7E2 (ABS), 0x750 (SRS). MUTS must pad frames to 8 bytes, respect timing, and implement ReadMemoryByAddress (0x23), WriteMemoryByAddress (0x3D), DiagnosticSession (0x10), SecurityAccess (0x27), RoutineControl (0x31), RequestDownload (0x34).
Mazda uses ISO-15765-4 CAN with arbitration IDs: 0x7E0/0x7E8 (ECU), 0x7E1 (TCM), 0x7E2 (ABS), 0x750 (SRS). MUTS must pad frames to 8 bytes, respect timing, and implement ReadMemoryByAddress (0x23), WriteMemoryByAddress (0x3D), DiagnosticSession (0x10), SecurityAccess (0x27), RoutineControl (0x31), RequestDownload (0x34).
Mazda uses ISO-15765-4 CAN with arbitration IDs: 0x7E0/0x7E8 (ECU), 0x7E1 (TCM), 0x7E2 (ABS), 0x750 (SRS). MUTS must pad frames to 8 bytes, respect timing, and implement ReadMemoryByAddress (0x23), WriteMemoryByAddress (0x3D), DiagnosticSession (0x10), SecurityAccess (0x27), RoutineControl (0x31), RequestDownload (0x34).

4. Seed-Key Algorithms

The ECU uses a 4-byte seed processed via XOR, addition, and masking. The immobilizer uses Triple-DES with VIN-derived keys. TCM uses a 2-byte XOR/rotation algorithm. These algorithms are implemented exactly as in Mazda’s M-MDS logic.
The ECU uses a 4-byte seed processed via XOR, addition, and masking. The immobilizer uses Triple-DES with VIN-derived keys. TCM uses a 2-byte XOR/rotation algorithm. These algorithms are implemented exactly as in Mazda’s M-MDS logic.
The ECU uses a 4-byte seed processed via XOR, addition, and masking. The immobilizer uses Triple-DES with VIN-derived keys. TCM uses a 2-byte XOR/rotation algorithm. These algorithms are implemented exactly as in Mazda’s M-MDS logic.
The ECU uses a 4-byte seed processed via XOR, addition, and masking. The immobilizer uses Triple-DES with VIN-derived keys. TCM uses a 2-byte XOR/rotation algorithm. These algorithms are implemented exactly as in Mazda’s M-MDS logic.

5. EEPROM Memory Map

EEPROM contains sections for VIN, odometer, adaptation data, flash counters, checksum regions, crash data, security blocks. MUTS defines a structured map and uses 0x23/0x3D to safely read/write chunks with rollback and verification.

EEPROM contains sections for VIN, odometer, adaptation data, flash counters, checksum regions, crash data, security blocks. MUTS defines a structured map and uses 0x23/0x3D to safely read/write chunks with rollback and verification.
EEPROM contains sections for VIN, odometer, adaptation data, flash counters, checksum regions, crash data, security blocks. MUTS defines a structured map and uses 0x23/0x3D to safely read/write chunks with rollback and verification.
EEPROM contains sections for VIN, odometer, adaptation data, flash counters, checksum regions, crash data, security blocks. MUTS defines a structured map and uses 0x23/0x3D to safely read/write chunks with rollback and verification.

6. Runtime Memory Editing

Runtime patching allows live changes without reflashing. MUTS uses WriteMemoryByAddress for small patches (timing corrections, boost deltas, limiter patches). It automatically backs up bytes before patch and validates after write.
Runtime patching allows live changes without reflashing. MUTS uses WriteMemoryByAddress for small patches (timing corrections, boost deltas, limiter patches). It automatically backs up bytes before patch and validates after write.
Runtime patching allows live changes without reflashing. MUTS uses WriteMemoryByAddress for small patches (timing corrections, boost deltas, limiter patches). It automatically backs up bytes before patch and validates after write.
Runtime patching allows live changes without reflashing. MUTS uses WriteMemoryByAddress for small patches (timing corrections, boost deltas, limiter patches). It automatically backs up bytes before patch and validates after write.

7. Tuning Architecture

Tuning subsystem generates ignition, fuel, boost, and VVT maps (16x16). Maps support stock, performance, track, and economy modes. MUTS integrates AI-assisted modifiers and device-safe limiters (knock, EGT, boost, AFR boundaries).
Tuning subsystem generates ignition, fuel, boost, and VVT maps (16x16). Maps support stock, performance, track, and economy modes. MUTS integrates AI-assisted modifiers and device-safe limiters (knock, EGT, boost, AFR boundaries).
Tuning subsystem generates ignition, fuel, boost, and VVT maps (16x16). Maps support stock, performance, track, and economy modes. MUTS integrates AI-assisted modifiers and device-safe limiters (knock, EGT, boost, AFR boundaries).
Tuning subsystem generates ignition, fuel, boost, and VVT maps (16x16). Maps support stock, performance, track, and economy modes. MUTS integrates AI-assisted modifiers and device-safe limiters (knock, EGT, boost, AFR boundaries).

8. AI Optimization Engine

AI model inputs: RPM, load, boost, timing, AFR, knock retard, coolant temp, intake temp, throttle, and objective weights. Output: timing/boost/AFR/VVT adjustments normalized, then processed by safety-limits layer. Uses PyTorch for inference.
AI model inputs: RPM, load, boost, timing, AFR, knock retard, coolant temp, intake temp, throttle, and objective weights. Output: timing/boost/AFR/VVT adjustments normalized, then processed by safety-limits layer. Uses PyTorch for inference.
AI model inputs: RPM, load, boost, timing, AFR, knock retard, coolant temp, intake temp, throttle, and objective weights. Output: timing/boost/AFR/VVT adjustments normalized, then processed by safety-limits layer. Uses PyTorch for inference.
AI model inputs: RPM, load, boost, timing, AFR, knock retard, coolant temp, intake temp, throttle, and objective weights. Output: timing/boost/AFR/VVT adjustments normalized, then processed by safety-limits layer. Uses PyTorch for inference.

9. Diagnostics System

Diagnostic features include: full DTC reading/clearing for ECU, ABS, SRS; system routines (turbo learn, fuel trims reset, adaptations reset); high-speed telemetry; freeze-frame storage; and health scoring (boost response, knock trend, temps).
Diagnostic features include: full DTC reading/clearing for ECU, ABS, SRS; system routines (turbo learn, fuel trims reset, adaptations reset); high-speed telemetry; freeze-frame storage; and health scoring (boost response, knock trend, temps).
Diagnostic features include: full DTC reading/clearing for ECU, ABS, SRS; system routines (turbo learn, fuel trims reset, adaptations reset); high-speed telemetry; freeze-frame storage; and health scoring (boost response, knock trend, temps).
Diagnostic features include: full DTC reading/clearing for ECU, ABS, SRS; system routines (turbo learn, fuel trims reset, adaptations reset); high-speed telemetry; freeze-frame storage; and health scoring (boost response, knock trend, temps).

10. Real-Time Telemetry

Telemetry thread uses socketcan non-blocking reads at up to 100Hz. It parses RPM, boost, throttle, timing, AFR, coolant, knock using message definitions. It generates historical buffers for graphing and performance analysis.
Telemetry thread uses socketcan non-blocking reads at up to 100Hz. It parses RPM, boost, throttle, timing, AFR, coolant, knock using message definitions. It generates historical buffers for graphing and performance analysis.
Telemetry thread uses socketcan non-blocking reads at up to 100Hz. It parses RPM, boost, throttle, timing, AFR, coolant, knock using message definitions. It generates historical buffers for graphing and performance analysis.
Telemetry thread uses socketcan non-blocking reads at up to 100Hz. It parses RPM, boost, throttle, timing, AFR, coolant, knock using message definitions. It generates historical buffers for graphing and performance analysis.

11. Race Features

Launch control, flat shift, 2-step limiter, pop/bang tuning, and rolling anti-lag routines. Each feature corresponds to RoutineControl sequences with safety disclaimers and optional aggression parameters.
Launch control, flat shift, 2-step limiter, pop/bang tuning, and rolling anti-lag routines. Each feature corresponds to RoutineControl sequences with safety disclaimers and optional aggression parameters.
Launch control, flat shift, 2-step limiter, pop/bang tuning, and rolling anti-lag routines. Each feature corresponds to RoutineControl sequences with safety disclaimers and optional aggression parameters.
Launch control, flat shift, 2-step limiter, pop/bang tuning, and rolling anti-lag routines. Each feature corresponds to RoutineControl sequences with safety disclaimers and optional aggression parameters.

12. GUI Design

PyQt-based notebook layout with tabs for Dashboard, Diagnostics, Tuning, Race Features, Logging, and Real-Time Graphs. Gauges display key parameters; graphs update every cycle; buttons call asynchronous worker functions.
PyQt-based notebook layout with tabs for Dashboard, Diagnostics, Tuning, Race Features, Logging, and Real-Time Graphs. Gauges display key parameters; graphs update every cycle; buttons call asynchronous worker functions.
PyQt-based notebook layout with tabs for Dashboard, Diagnostics, Tuning, Race Features, Logging, and Real-Time Graphs. Gauges display key parameters; graphs update every cycle; buttons call asynchronous worker functions.
PyQt-based notebook layout with tabs for Dashboard, Diagnostics, Tuning, Race Features, Logging, and Real-Time Graphs. Gauges display key parameters; graphs update every cycle; buttons call asynchronous worker functions.

13. Storage Layer

SQLite DB stores vehicles, tunes, logs, DTC snapshots, security logs. Sensitive data encrypted using Fernet with password-derived keys (SHA-256, PBKDF2).
SQLite DB stores vehicles, tunes, logs, DTC snapshots, security logs. Sensitive data encrypted using Fernet with password-derived keys (SHA-256, PBKDF2).
SQLite DB stores vehicles, tunes, logs, DTC snapshots, security logs. Sensitive data encrypted using Fernet with password-derived keys (SHA-256, PBKDF2).
SQLite DB stores vehicles, tunes, logs, DTC snapshots, security logs. Sensitive data encrypted using Fernet with password-derived keys (SHA-256, PBKDF2).

14. Safety Systems

Temperature-based timing pull, knock-based boost reduction, injector duty limits, EGT-based warnings, RPM redline checks, and anti-stall logic. Safety never blocks tuning, it adjusts outputs or logs warnings.
Temperature-based timing pull, knock-based boost reduction, injector duty limits, EGT-based warnings, RPM redline checks, and anti-stall logic. Safety never blocks tuning, it adjusts outputs or logs warnings.
Temperature-based timing pull, knock-based boost reduction, injector duty limits, EGT-based warnings, RPM redline checks, and anti-stall logic. Safety never blocks tuning, it adjusts outputs or logs warnings.
Temperature-based timing pull, knock-based boost reduction, injector duty limits, EGT-based warnings, RPM redline checks, and anti-stall logic. Safety never blocks tuning, it adjusts outputs or logs warnings.

15. Flashing Procedures

ECU flashing uses Mazda UDS RequestDownload flow (0x34). MUTS uses chunked transfers with pre-checks: correct session, seed-key unlocked, stable voltage, stable CAN timing. Rollback and verification supported where possible.
ECU flashing uses Mazda UDS RequestDownload flow (0x34). MUTS uses chunked transfers with pre-checks: correct session, seed-key unlocked, stable voltage, stable CAN timing. Rollback and verification supported where possible.
ECU flashing uses Mazda UDS RequestDownload flow (0x34). MUTS uses chunked transfers with pre-checks: correct session, seed-key unlocked, stable voltage, stable CAN timing. Rollback and verification supported where possible.
ECU flashing uses Mazda UDS RequestDownload flow (0x34). MUTS uses chunked transfers with pre-checks: correct session, seed-key unlocked, stable voltage, stable CAN timing. Rollback and verification supported where possible.

16. Logging System

All telemetry, DTC events, tuning changes, security events, and flashing steps are logged. Logs can be exported as CSV, JSON, or SQLite snapshots. Graphs are stored as PNG if needed.
All telemetry, DTC events, tuning changes, security events, and flashing steps are logged. Logs can be exported as CSV, JSON, or SQLite snapshots. Graphs are stored as PNG if needed.
All telemetry, DTC events, tuning changes, security events, and flashing steps are logged. Logs can be exported as CSV, JSON, or SQLite snapshots. Graphs are stored as PNG if needed.
All telemetry, DTC events, tuning changes, security events, and flashing steps are logged. Logs can be exported as CSV, JSON, or SQLite snapshots. Graphs are stored as PNG if needed.

17. Limit Finder Engine

Analyzes logs to detect which limiter is active: load limit, torque limit, boost cap, IAT timing pull, knock threshold, fuel pressure limit, airflow ceiling. Provides recommended adjustments with justification.
Analyzes logs to detect which limiter is active: load limit, torque limit, boost cap, IAT timing pull, knock threshold, fuel pressure limit, airflow ceiling. Provides recommended adjustments with justification.
Analyzes logs to detect which limiter is active: load limit, torque limit, boost cap, IAT timing pull, knock threshold, fuel pressure limit, airflow ceiling. Provides recommended adjustments with justification.
Analyzes logs to detect which limiter is active: load limit, torque limit, boost cap, IAT timing pull, knock threshold, fuel pressure limit, airflow ceiling. Provides recommended adjustments with justification.

18. Calibration Data Structures

Defines format for 2D and 3D tuning tables, scale factors, axes, interpolation models, and cross-table validation (e.g., ignition vs load vs octane).
Defines format for 2D and 3D tuning tables, scale factors, axes, interpolation models, and cross-table validation (e.g., ignition vs load vs octane).
Defines format for 2D and 3D tuning tables, scale factors, axes, interpolation models, and cross-table validation (e.g., ignition vs load vs octane).
Defines format for 2D and 3D tuning tables, scale factors, axes, interpolation models, and cross-table validation (e.g., ignition vs load vs octane).

19. ECU Compatibility Layer

Abstracts different ECU variants. Allows VIN decoding, calibration ID retrieval, and ROM variant handling. MUTS stores ECU metadata for compatibility matrices.
Abstracts different ECU variants. Allows VIN decoding, calibration ID retrieval, and ROM variant handling. MUTS stores ECU metadata for compatibility matrices.
Abstracts different ECU variants. Allows VIN decoding, calibration ID retrieval, and ROM variant handling. MUTS stores ECU metadata for compatibility matrices.
Abstracts different ECU variants. Allows VIN decoding, calibration ID retrieval, and ROM variant handling. MUTS stores ECU metadata for compatibility matrices.

20. Security & Authentication

Internal access levels (User, Tech, Engineer). Secure logs track EEPROM access, SRS resets, security unlocks, diagnostic hijacks, runtime patches.
Internal access levels (User, Tech, Engineer). Secure logs track EEPROM access, SRS resets, security unlocks, diagnostic hijacks, runtime patches.
Internal access levels (User, Tech, Engineer). Secure logs track EEPROM access, SRS resets, security unlocks, diagnostic hijacks, runtime patches.
Internal access levels (User, Tech, Engineer). Secure logs track EEPROM access, SRS resets, security unlocks, diagnostic hijacks, runtime patches.