Server Code Review: SOLID, DRY, KISS Principles

Date: 2026-01-28 **Scope:** WebSocket handlers and message routing system

Executive Summary

The refactored code demonstrates good adherence to SOLID principles but has notable DRY violations that should be addressed. KISS is generally followed with appropriate complexity given the NN pipeline requirements.

SOLID Principles Analysis

Single Responsibility Principle (SRP) - PASS

Each handler has a clear, focused responsibility:

File	Responsibility
`MessageHandler`	Facade/coordinator for message processing
`MessageRouter`	Routes messages to appropriate handlers
`ObservationHandler`	Processes NN observations and spawn decisions
`TrainingHandler`	Manages training lifecycle
`GateHandler`	Handles simulation gate statistics
`SystemHandler`	System health and status
`GameStateHandler`	Game events (death, success, outcome)

Open/Closed Principle (OCP) - PASS

- Plugin-style handler registration allows extension without modification
- `router.register()` pattern enables adding new message types without changing existing code
- Handler classes can be extended or replaced independently

Liskov Substitution Principle (LSP) - N/A

No inheritance hierarchy in the handler system. Each handler is a standalone class.

Interface Segregation Principle (ISP) - PASS

- Handlers receive only the dependencies they need
- No fat interfaces forcing handlers to implement unused methods
- Callable type hints for getter functions (e.g., `thinking_stats_getter`)

Dependency Inversion Principle (DIP) - PASS

- Handlers depend on abstractions via `TYPE_CHECKING` imports
- All dependencies injected through constructors
- No hard-coded instantiation within handlers

DRY Violations

1. CRITICAL: `_create_error_response()` Duplicated 6 Times

Location: All handler files + message_handler.py

File	Lines	Variant
`message_handler.py`	313-329	Extended (includes retryable, supportedMessageTypes)
`observation_handler.py`	557-567	Standard
`training_handler.py`	208-222	Standard
`gate_handler.py`	142-156	Standard
`system_handler.py`	260-274	Standard
`game_state_handler.py`	179-193	Standard

Impact:

- 5 identical implementations + 1 extended variant
- ~60 lines of duplicated code
- Risk of inconsistent error response formats

Solution: Extract to a shared utility module.

2. MODERATE: Component Initialization Pattern in `_init_components()`

Location: `message_handler.py` lines 79-131

Pattern repeated 6 times:

```
self.component = None
try:
    self.component = Component()
    logger.info("Component initialized")
except Exception as e:
    logger.warning(f"Failed to initialize Component: {e}")
```

Impact:

- Repetitive boilerplate code
- 50+ lines that could be reduced to ~15

Solution: Create a factory helper function.

KISS Analysis

Appropriate Complexity

1. `ObservationHandler._process_observation()` - Complex but justified
   - Handles multi-stage NN pipeline (preprocess → feature extraction → inference → gate → buffer)
   - Well-decomposed into smaller methods
2. `MessageRouter.route()` **vs** `route_raw()` - Acceptable
   - Two methods serve different use cases (ParsedMessage vs raw dict)
   - Backward compatibility requirement justifies duplication

Potential Simplifications

1. **Error handling in route_raw()** - Could consolidate TypeError catch with general Exception

---

Recommendations

Priority 1: Fix DRY Violations

1. Create `websocket/handlers/base.py` with shared error response utility
2. Refactor all handlers to use the shared utility
3. Consider base class for common handler functionality

Priority 2: Simplify Initialization

1. Create component factory helper in `message_handler.py`
2. Use dictionary mapping for cleaner initialization

Checklist

- [x] SRP: Each class has single responsibility
- [x] OCP: Open for extension, closed for modification
- [x] LSP: N/A (no inheritance)
- [x] ISP: No fat interfaces
- [x] DIP: Dependencies injected
- [x] DRY: Error response duplicated - **FIXED**
- [ ] DRY: Init pattern duplicated (Low priority - acceptable boilerplate)
- [x] KISS: Complexity appropriate for requirements

Fixes Applied (2026-01-28)

DRY Violation #1: Error Response Duplication - FIXED

Created: `websocket/handlers/base.py`

- Shared `create_error_response()` function
- `RETRYABLE_ERROR_CODES` constant
- `is_retryable_error()` helper

Updated files to use shared utility:

1. `message_handler.py` - uses extended params (retryable, supported_message_types)
2. `observation_handler.py` - removed duplicate method
3. `training_handler.py` - removed duplicate method
4. `gate_handler.py` - removed duplicate method
5. `system_handler.py` - removed duplicate method
6. `game_state_handler.py` - removed duplicate method

Result: ~60 lines of duplicated code eliminated. All 25 unit tests pass.

Remaining Action Items

1. ~~Create `websocket/handlers/base.py` with `create_error_response()` utility~~ ☑ DONE
2. ~~Update all 6 files to use shared utility~~ ☑ DONE
3. Component initialization helper (Low priority - current pattern is readable)