# **DRY Improvement Recommendations for Financial Modeling Codebase**

After analyzing the codebase, I've identified several areas where the code could be improved to follow the DRY (Don't Repeat Yourself) principle more closely:

## **1. Centralize LLM API Calls**

There are multiple similar implementations of LLM API calls across different files:

* In fm\_utils.py, user\_base\_formula.py, and revenue\_formula.py, there are repetitive patterns for making API calls to OpenAI and Anthropic.
* The execute\_llm\_step function in user\_base\_formula.py could be moved to a common utility module and reused across all files.

**Recommendation:** Create a dedicated llm\_client.py module with standardized functions for different types of LLM interactions.

## **2. Standardize Prompt Generation**

The codebase contains numerous prompt-generating functions with similar patterns:

* user\_base\_prompt\_step1, user\_base\_prompt\_step2, etc. in user\_base\_formula.py
* revenue\_prompt\_step1, revenue\_prompt\_step2, etc. in revenue\_formula.py
* Various prompt functions in fm\_utils.py

**Recommendation:** Create a template-based prompt generation system with a common structure and customizable components.

## **3. Consolidate Schema Definitions**

The schemas/string.py file contains multiple similar schema classes:

* StringSchema, StringSchema2, and StringSchema3 have overlapping fields
* Several schemas share common patterns but are defined separately

**Recommendation:** Implement inheritance for related schemas and use composition for common field patterns.

## **4. Abstract Common API Endpoint Patterns**

In main.py, there are repetitive patterns for API endpoints:

* Many endpoints follow the pattern of receiving input, calling a processing function, and returning results
* Error handling is duplicated across endpoints

**Recommendation:** Create decorator functions or middleware to handle common patterns like validation, error handling, and response formatting.

## **5. Refactor Step-Based Processing Logic**

Both user\_base\_formula.py and revenue\_formula.py implement multi-step processing with similar patterns:

* Each has step1, step2, etc. functions with corresponding prompt functions
* The execution flow is similar across both modules

**Recommendation:** Create a generic step processor that can be configured with specific step functions and prompts.

## **6. Implement Common Error Handling**

Error handling is inconsistent and duplicated:

* Some functions have try/except blocks, others don't
* The error response format varies across functions

**Recommendation:** Standardize error handling with a central error handling utility that provides consistent error responses.

## **7. Extract Common Data Transformation Logic**

There are repeated patterns for data transformation:

* Functions like parse\_final\_analysis, extract\_answer, and extract\_variables perform similar text parsing operations
* Data structure conversions are duplicated across files

**Recommendation:** Create utility functions for common data transformation operations.

## **8. Implement Configuration Management**

API keys and other configuration values are hardcoded in multiple places:

* Anthropic API key is directly in the code in revenue\_formula.py

**Recommendation:** Move all configuration to a central configuration management system, possibly extending the existing app/core/config.py.

## **9. Create Reusable Testing Utilities**

While not directly visible in the code, testing likely involves similar setup and teardown operations across different modules.

**Recommendation:** Create common testing utilities and fixtures to avoid duplication in test code.

## **10. Standardize Async/Sync Function Patterns**

The codebase mixes async and sync functions inconsistently:

* Some functions are async but don't need to be
* Similar functions have different async/sync patterns

**Recommendation:** Establish clear guidelines for when to use async functions and standardize the pattern across the codebase.

Implementing these recommendations would significantly reduce code duplication, improve maintainability, and make the codebase more robust and easier to extend.