Piotr Copek
Zuzanna Micorek
Hyunseok Cho
Szymon Molicki
Mateusz Znaleźniak

31.01.2026

Structure Modeling and prototype

Software Engineering

Index

1. Structure Modeling - Class Model Preparation

Subjects (Active entities that perform operations)

Term	Description	Class Candidate
SpamService	Service managing spam classification requests	+
SpamChecker	Classifier performing spam detection	+
GeneralRiskOverseer	Alternative classifier using sliding window	+
Logger	Logging system managing application logs	+
Flask Application	Web application managing HTTP requests	+

Conceptual Entities (Data structures and configurations)

Term	Description	Class Candidate
SpamPrediction	Result of spam classification	+
GroPrediction	Result from sliding window classifier	+
GroWindowConfig	Configuration for sliding window processing	+
GroTrainConfig	Configuration for model training	+

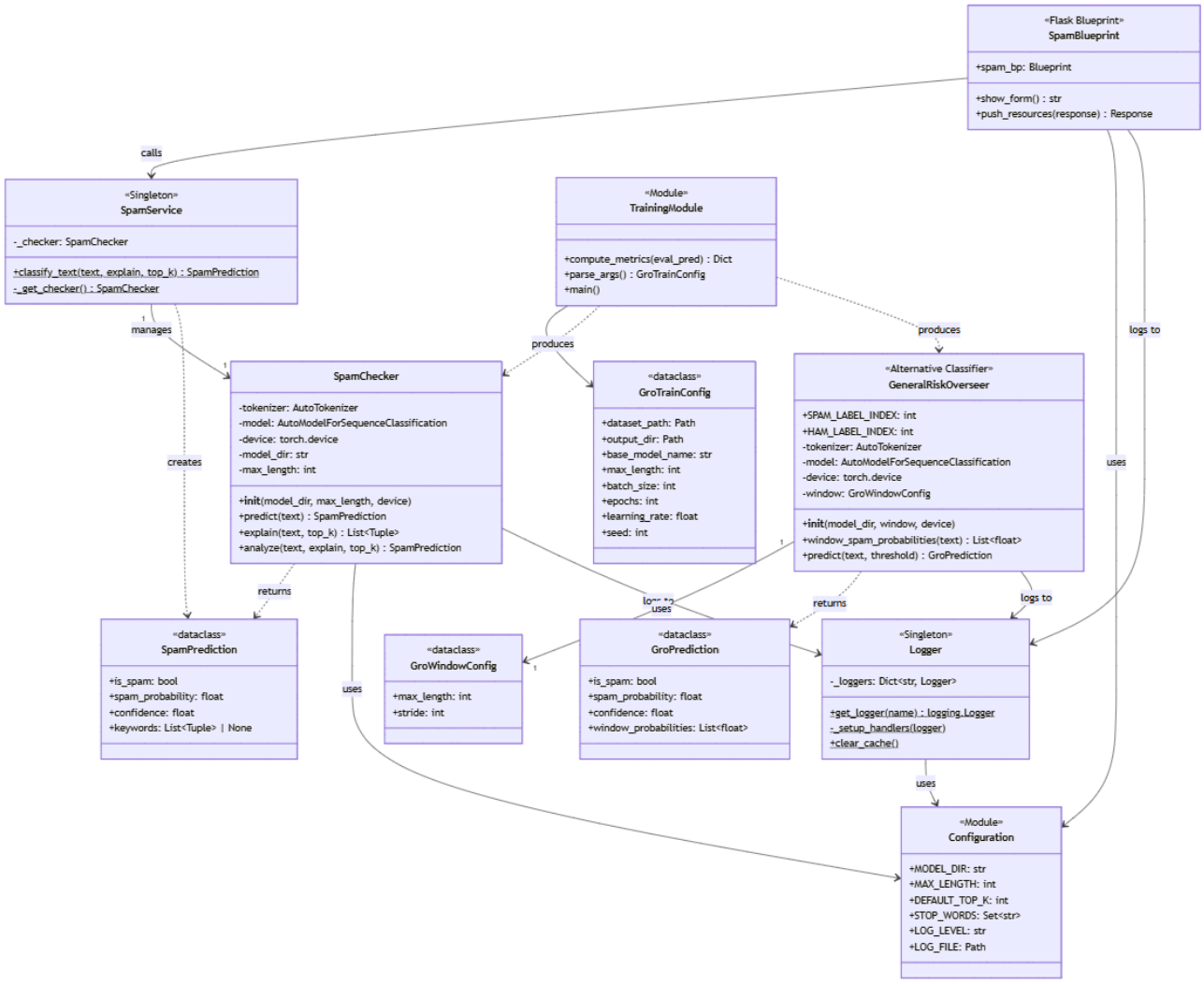
Persons (User roles and actors)

Term	Description	Class Candidate
User	Person submitting email text for analysis	- (Actor, not stored)
Administrator	Person managing the application	- (Out of scope)

Note: In this project, we don't store user data or activity logs. Users interact with the system anonymously, and no database is required. For maintainability purposes we introduced a simple file-based logging system to track application events and errors.

2. UML Class Diagram

2.1 Main Classes with Associations



2.2 Generalizations

This project uses **composition over inheritance** principles. There are no class hierarchies or generalizations. Instead, we use:

- **Singleton pattern** (SpamService, Logger): Ensures single instances of critical components
- **Facade pattern** (SpamService): Simplifies complex subsystem interactions
- **Dataclass pattern** (SpamPrediction, GroPrediction, configurations): Provides immutable,

type-safe data structures

2.3 Advantages of Modular Structure

The modular architecture provides such benefits:

1. Separation of Concerns

- Each module handles one responsibility (services, routing, logging, training)
- Changes in one module rarely affect others
- Easier to understand code

2. Reusability

- Modules can be imported and used independently
- `SpamChecker` can be used in other projects without Flask dependency
- `Logger` is generic and applicable across any Python application

3. Maintainability

- Bugs are isolated to specific modules
- Updates to one component don't break others
- Clear module boundaries make debugging faster
- Configuration centralized in `config.py`

4. Testability

- Mock dependencies easily in unit tests
- Test modules independently without full application
- Integration tests verify module interactions

5. Scalability

- Add new classifiers without modifying existing code
- Extend routes without touching service layer
- New team members can work on isolated modules

6. Flexibility

- Replace `SpamChecker` with `GeneralRiskOverseer` without changing Flask routes

- Swap logging backends in one place
- Update configuration without code changes

3. Why No Database?

3.1 Principle

Our spam detection system is designed as a stateless service with the following characteristics:

1. **No User Accounts**: Users access the system anonymously
2. **No Activity Logging**: We don't track user behavior or submissions
3. **No Persistent Storage**: Each request is independent
4. **Privacy-First**: No user data is stored or retained as emails may contain fragile information

3.2 Data Flow

```
User Input
    ↓
Flask Request Handler
    ↓
SpamService (in-memory)
    ↓
Model Inference (in-memory)
    ↓
Result Returned to User
    ↓
No data persisted
```

3.3 What IS Stored

- **Model Files**: Pre-trained models (read-only on filesystem)
- **Application Logs**: System events and errors (for debugging, in log files)
- **Configuration**: Application settings (in `config.py`)

None of this requires the database.

4. Prototype Development

4.1 Project Structure

```
spam_detection/
├── api/
│   ├── __init__.py
│   └── routes_spam.py          # Flask Blueprint / Endpoints
├── services/
│   ├── __init__.py
│   ├── spam_checker.py         # SpamService & SpamChecker classes
│   └── gro.py                  # GeneralRiskOverseer class
├── core/
│   ├── __init__.py
│   ├── config.py               # Configuration module
│   └── logger.py               # Logger class
├── training/
│   ├── __init__.py
│   └── train.py                # TrainingModule
├── templates/
│   ├── home.html               # Landing page
│   ├── check_spam.html         # Results page
│   └── 404.html                # Error page
├── static/
│   └── css/
│       └── styles.css          # Styling
├── models/
│   └── gro_spam_v1/            # Model files (not in repo)
├── logs/
│   ├── app.log                 # Application logs
│   └── errors.log              # Error logs
├── main.py                     # Flask app factory
└── __main__.py                 # Entry point
```

4.2 Server-Side Modules (Python)

`main.py` - Application Factory

```python
"""Flask application entry point"""

def create_app() -> Flask:
    """Creates and configures Flask application"""
    app = Flask(__name__)

    @app.route("/")
    def home():
        """Home page route"""
        return render_template("home.html")

    app.register_blueprint(spam_bp)
    return app
```

`routes_spam.py` - API Endpoints

```python
"""Spam detection API routes"""

spam_bp = Blueprint("spam", __name__, url_prefix="/")

@spam_bp.route("/form", methods=["GET", "POST"])
def show_form():
    """Handle spam detection form display and processing"""
    # GET: Display form
    # POST: Process spam check with retry logic
    pass

@spam_bp.after_request
def push_resources(response):
    """Add resource preload headers for optimization"""
    pass
```

`spam_checker.py` - Spam Detection Service

```python
"""Core spam detection logic"""

class SpamService:
    """Singleton facade for spam classification"""
    _checker: SpamChecker | None = None

    @classmethod
    def classify_text(cls, text, explain, top_k) -> SpamPrediction:
        """Main entry point for spam detection"""
        pass

    @classmethod
    def _get_checker(cls) -> SpamChecker:
        """Retrieve or create SpamChecker instance"""
        pass

class SpamChecker:
    """BERT-based spam classifier with explanations"""

    def __init__(self, model_dir, max_length, device):
        """Initialize model and tokenizer"""
        pass

    def predict(self, text) -> SpamPrediction:
        """Binary spam classification"""
        pass

    def explain(self, text, top_k) -> List[Tuple]:
        """Extract top-k spam keywords using IntegratedGradients"""
        pass

    def analyze(self, text, explain, top_k) -> SpamPrediction:
        """Combined prediction + explanation"""
        pass
```

`gro.py` - Alternative Classifier

Sliding window processing, probability aggregation.

```
"""GeneralRiskOverseer - sliding window classifier"""

class GeneralRiskOverseer:
    """BERT-based classifier using sliding window approach"""

    def __init__(self, model_dir, window, device):
        """Initialize with window configuration"""
        pass

    def window_spam_probabilities(self, text) -> List[float]:
        """Calculate spam probability for each window"""
        pass

    def predict(self, text, threshold) -> GroPrediction:
        """Aggregate window predictions"""
        pass
```

`logger.py` - Logging System

```python
"""Professional logging infrastructure"""

class Logger:
    """Singleton logger with file rotation and error separation"""
    _loggers: Dict[str, logging.Logger] = {}

    @classmethod
    def get_logger(cls, name) -> logging.Logger:
        """Get or create logger instance"""
        pass

    @classmethod
    def _setup_handlers(cls, logger):
        """Configure file and console handlers"""
        pass

    @classmethod
    def clear_cache(cls):
        """Clear logger cache (for testing)"""
        pass
```

`config.py` - Configuration

```
"""Application-wide configuration constants"""

# Model Configuration
MODEL_DIR: str
MAX_LENGTH: int
DEFAULT_TOP_K: int
STOP_WORDS: Set[str]

# Logging Configuration
LOG_DIR: Path
LOG_LEVEL: str
LOG_FORMAT: str
LOG_FILE: Path
ERROR_LOG_FILE: Path
```

`train.py` - Training Pipeline

```
"""Model training orchestration"""

def compute_metrics(eval_pred) -> Dict[str, float]:
    """Calculate accuracy, precision, recall, F1"""
    pass

def parse_args() -> GroTrainConfig:
    """Parse CLI training arguments"""
    pass

def main():
    """Execute complete training pipeline"""
    # 1. Load dataset
    # 2. Tokenize text
    # 3. Train BERT model
    # 4. Evaluate on test set
    # 5. Save model checkpoint
    pass
```

4.3 Client-Side Modules (HTML/JavaScript)

`home.html` - Landing page with spam detection form

- Email text input area
- Submit button
- Theme toggle (light/dark)
- Font size toggle
- Responsive design

`check_spam.html` - Display spam detection results

- Spam/Not Spam verdict
- Confidence indicator
- Top keywords (if spam)
- Original email text
- "Check another email" button

`404.html` - Error page for service failures

- User-friendly error message
- "Return to Home" button
- Consistent styling with theme support

Client-Side JavaScript Functions

```
// Theme toggle
function toggleTheme() {
    // Switch between light and dark mode
    // Save preference to localStorage
}

// Font size toggle
function toggleFont() {
    // Cycle through small/medium/large
    // Save preference to localStorage
}

// On page load
window.addEventListener('DOMContentLoaded', function() {
    // Restore saved theme and font preferences
});
```

4.4 Data Classes

`SpamPrediction`

```
@dataclass(frozen=True)
class SpamPrediction:
    is_spam: bool
    spam_probability: float
    confidence: float
    keywords: List[Tuple[str, float]] | None = None
```

`GroPrediction`

```python
@dataclass(frozen=True)
class GroPrediction:
    is_spam: bool
    spam_probability: float
    confidence: float
    window_probabilities: List[float]
```

`GroWindowConfig`

```python
@dataclass(frozen=True)
class GroWindowConfig:
    max_length: int = 256
    stride: int = 96
```

`GroTrainConfig`

```python
@dataclass(frozen=True)
class GroTrainConfig:
    dataset_path: Path
    output_dir: Path
    base_model_name: str
    max_length: int
    batch_size: int
    epochs: int
    learning_rate: float
    seed: int
```

5. External Dependencies

Library	Version	Purpose
Flask	2.x	Web framework
transformers	Latest	BERT models

Library	Version	Purpose
torch	Latest	Neural networks
captum	Latest	Model interpretability
datasets	Latest	Dataset loading
evaluate	Latest	Metric computation

6. Testing Prototype

6.1 Manual Testing Scenarios

Basic Spam Detection

1. Navigate to home page
2. Enter spam text: "CONGRATULATIONS! You've WON $1000000! Click HERE now!!!"
3. Detection of spam.

Legitimate Email Detection

1. Navigate to home page
2. Enter normal text: "Hi John, Let's meet tomorrow at 3pm to discuss the project."
3. Detection of legitimate email.

Theme Toggle

1. Click theme toggle button
2. Theme changes without reloading the page

Error Handling

1. Stop the ML model service
2. On submit page `404.html` loads

7. Conclusion

7.1 Prototype

We have successfully developed prototype of the Email Spam Detection Web Application. The prototype includes:

- Complete server-side implementation
- Full client-side interface
- Model training pipeline
- Error handling
- Accessibility features

7.2 Why No Database

The system is designed as a stateless, privacy-first service:

- No user accounts or authentication
- No activity logging or tracking
- Each request is independent
- No data persistence required

All necessary data is stored on the filesystem, not in a database.

The prototype demonstrates a clean, modular architecture following object-oriented design principles without requiring a database layer.