Piotr Copek
Zuzanna Micorek
Hyunseok Cho
Szymon Molicki
Mateusz Znaleźniak

31.01.2026

Behavioral Modelling - Prototype #2

Software Engineering

Index

1. Introduction

Document presents the behavioral modeling for the Email Spam Detection System as part of Prototype #2 development. Building upon Prototype #1, which established the basic web interface and spam detection capability, Prototype #2 extends the system with:

- **Detailed interaction modeling** through sequence diagrams for the most complex use cases
- **Enhanced error handling** with retry logic and exponential backoff
- **Keyword explanations** using IntegratedGradients attribution
- **Logging** for debugging and monitoring
- **Activity flow diagrams** illustrating decision logic and process workflows

Behavioral modeling adheres to the following principles:

- **Privacy-First**: No data persistence, completely anonymous operations
- **Stateless Architecture**: No user accounts, sessions, or databases
- **Transparency**: Clear interaction flows with explainable AI results
- **Resilience**: Robust error handling with automatic retry mechanisms

2. Interaction Diagram Preparation

To determine which use cases require detailed sequence diagrams, we analyzed each use case based on the actual implementation:

Use Case	Actors	System Components	External Libraries	Complexity Score	Selected
Submit Email for Spam Analysis	1	6 (Blueprint, Service, Checker, Model, Tokenizer, Logger)	2 (Transformers, PyTorch)	**High (8)**	**+**
View Spam	1	2 (Blueprint,	1 (Flask)	Low (3)	**-**

Use Case	Actors	System Components	External Libraries	Complexity Score	Selected
Classification Result		Template)			
View Spam Explanation with Keywords	1	7 (Blueprint, Service, Checker, Model, Tokenizer, Captum, Logger)	3 (Transformers, PyTorch, Captum)	**Very High (10)**	**+**

Complexity Scoring:

- Each actor: +1 point
- Each system component: +1 point
- Each external library: +1 point
- Complex logic (retry, attribution): +2 points

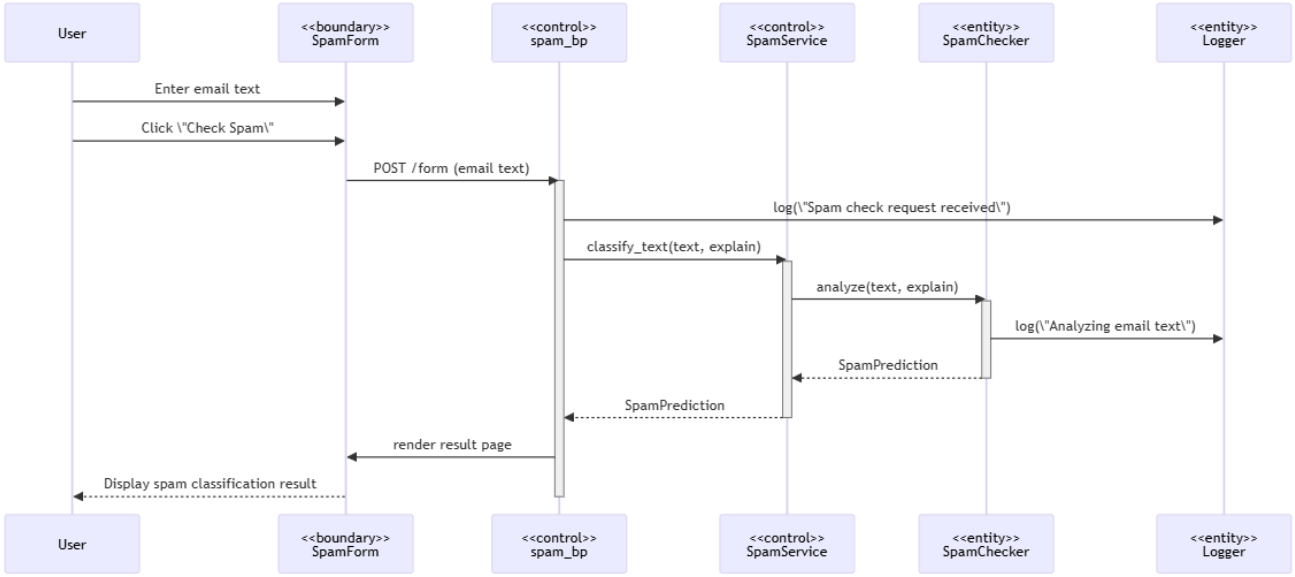
Selected for Detailed Modeling:

1. **Submit Email for Spam Analysis** (Complexity: 8) - Multi-component interaction with retry logic
2. **View Spam Explanation with Keywords** (Complexity: 10) - Advanced ML interpretability with IntegratedGradients

3. Sequence Diagrams for Complex Use Cases

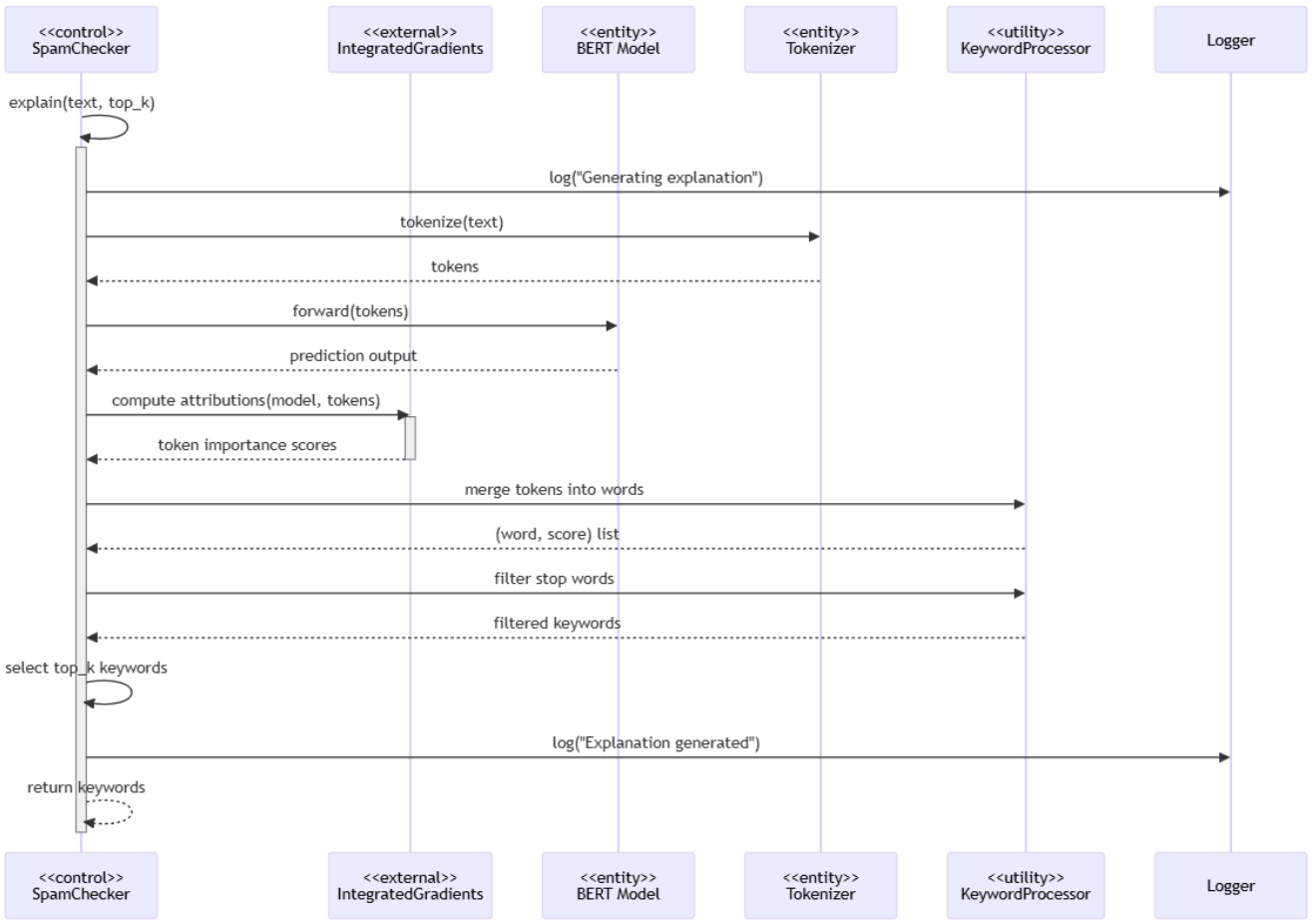
3.1 Submit Email for Spam Analysis with Retry Logic

Sequence diagram illustrates the complete flow from user submission through retry logic, model inference, and result rendering. This corresponds to the `show_form()` method in `routes_spam.py`.



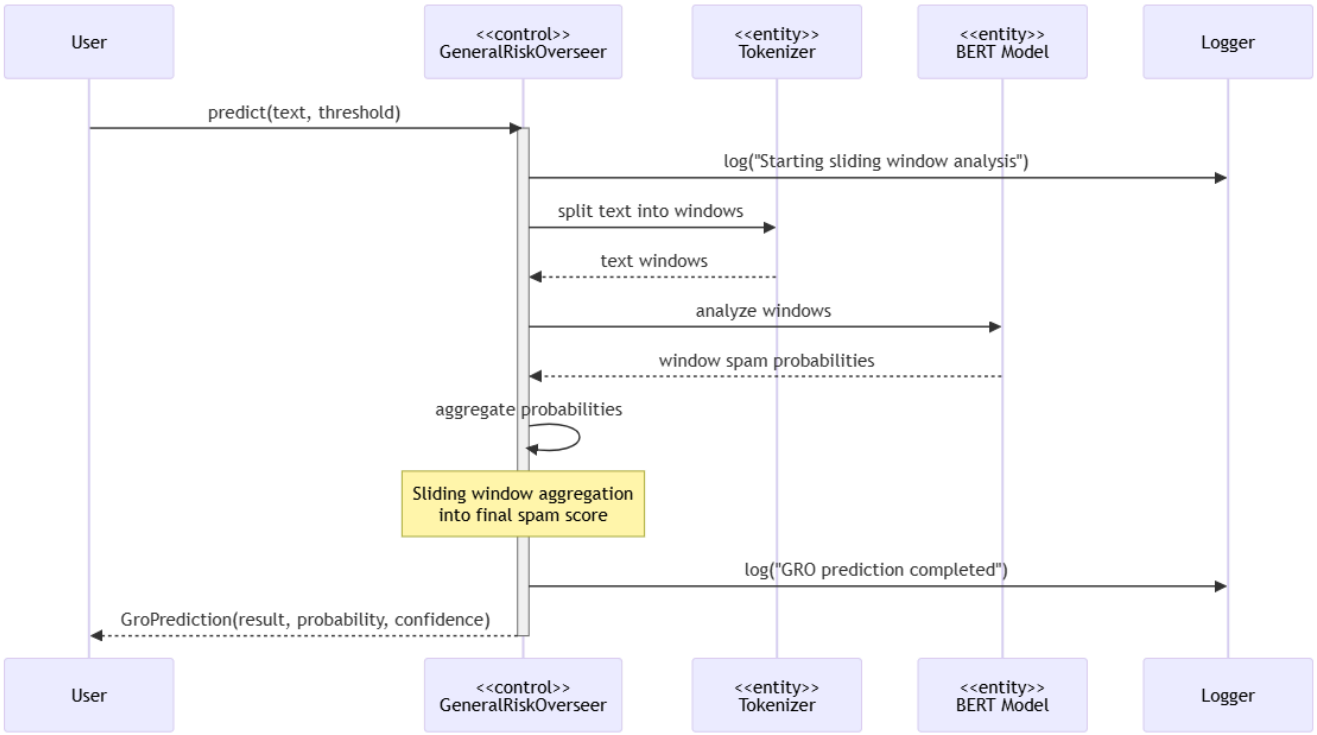
3.2 View Spam Explanation with Keywords (IntegratedGradients)

Sequence diagram shows the detailed attribution calculation process when explanation is requested. This corresponds to the `explain()` method in `spam_checker.py`.



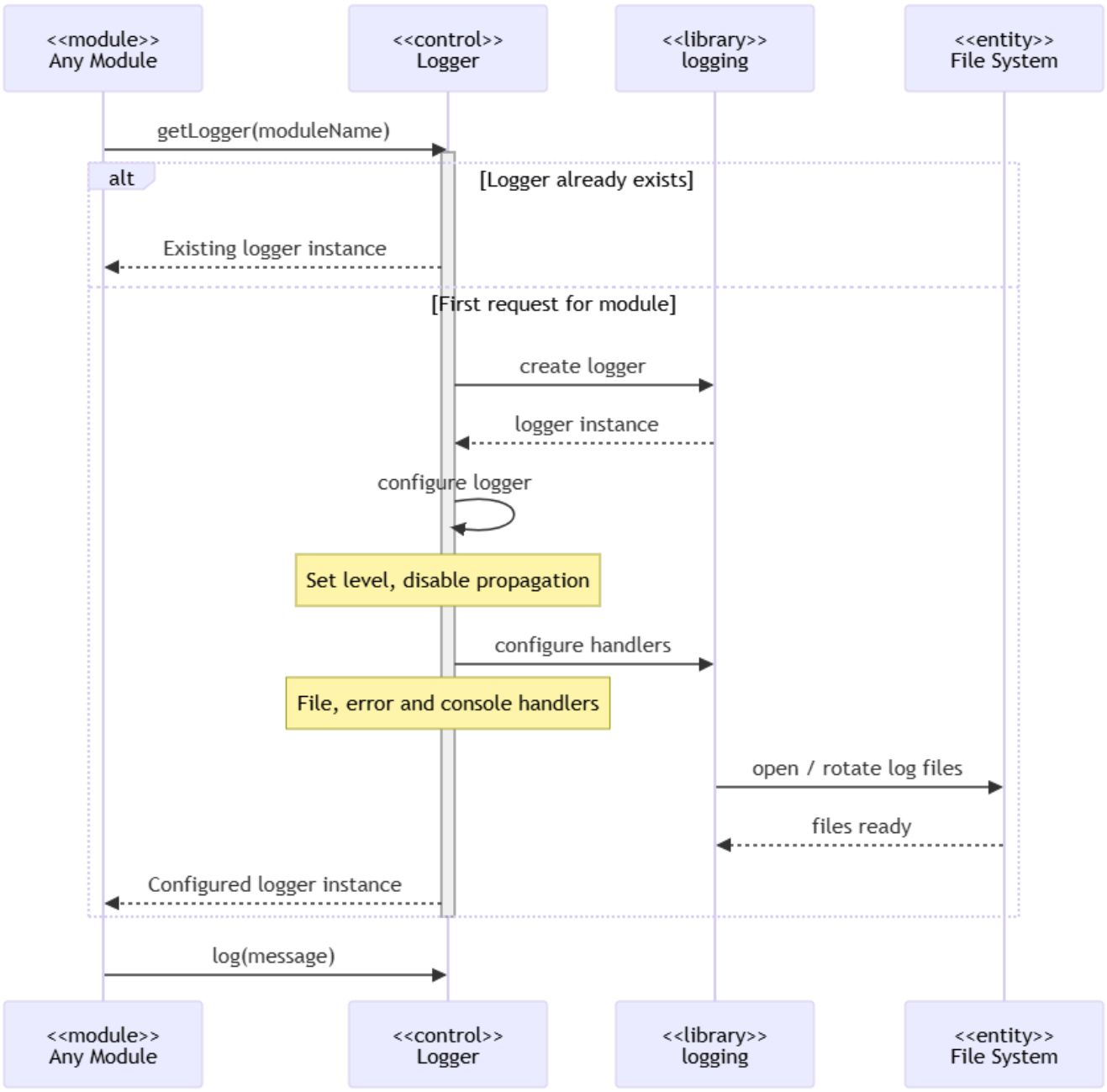
3.3 Alternative: GeneralRiskOverseer Sliding Window Classification

Sequence diagram for the sliding window classifier approach (from `gro.py`):



3.4 Logger Initialization and Singleton Pattern

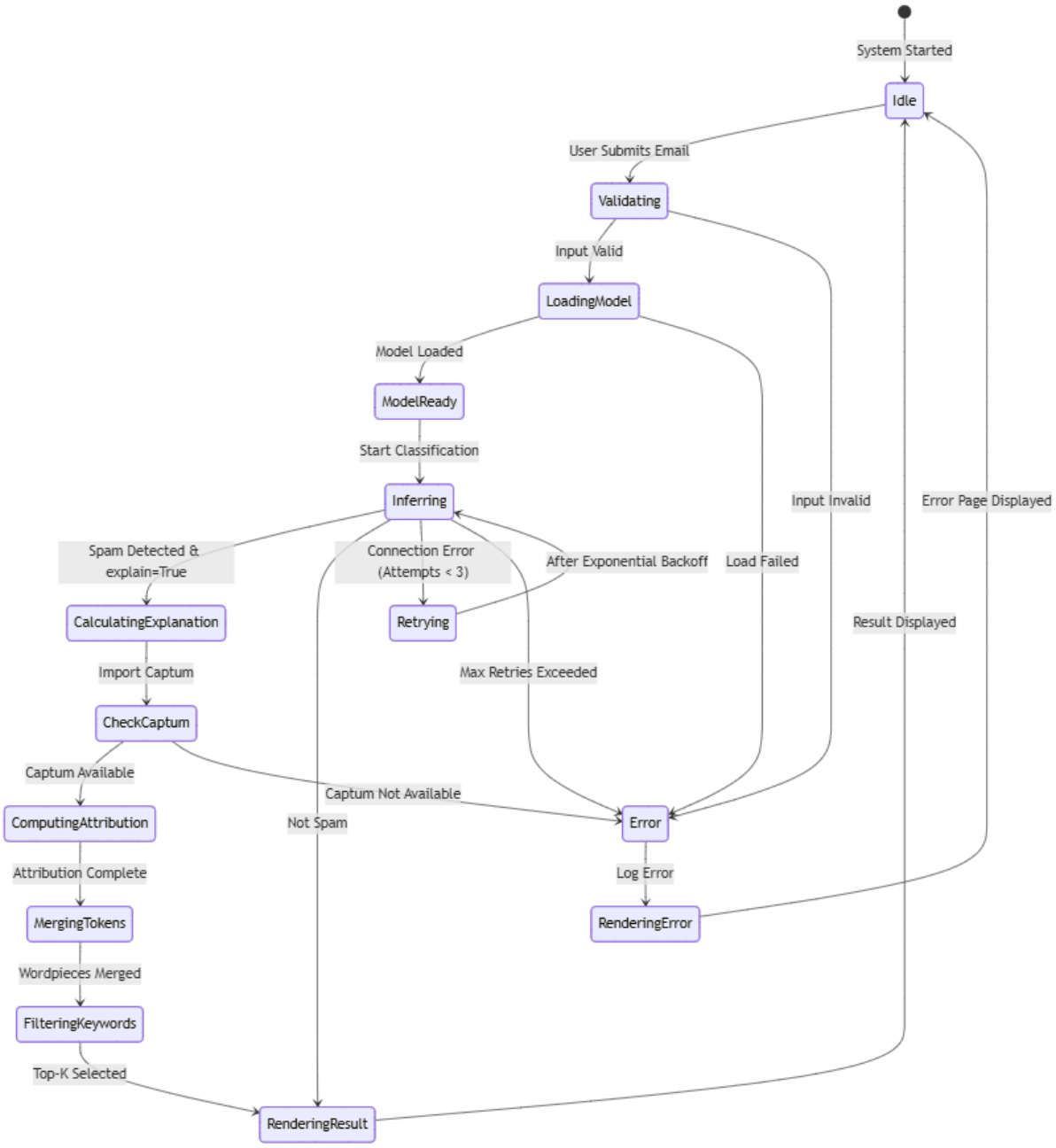
This diagram shows the logging infrastructure initialization from `logger.py`:



4. State Diagrams

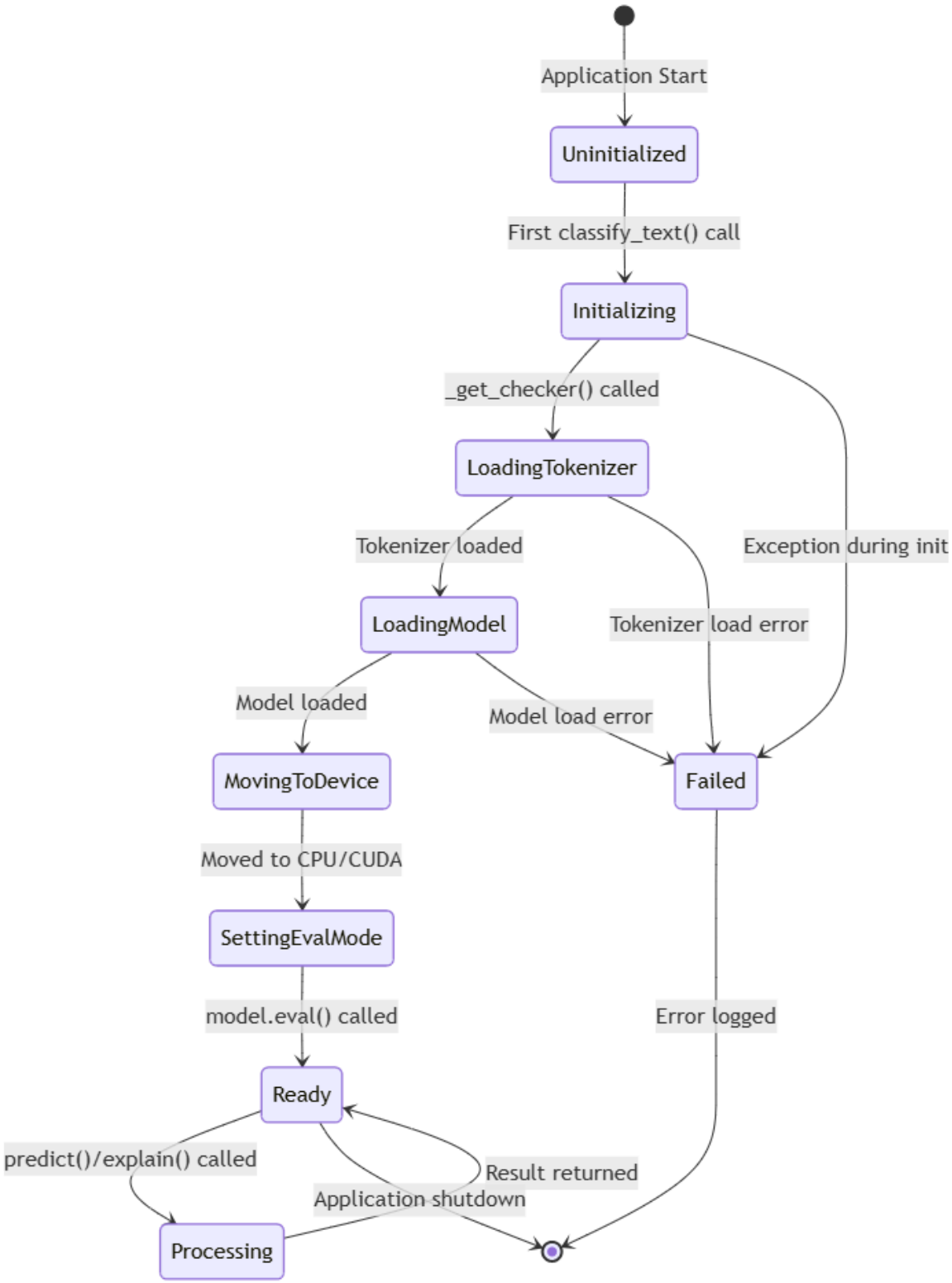
4.1 Classification Request State Machine

This state diagram shows the lifecycle of a spam classification request from submission to completion.



4.2 SpamChecker Object Lifecycle

This state diagram shows the lifecycle of the `SpamChecker` singleton instance:



5. Development

5.1 Extensions from Prototype #1

Prototype #2 builds upon Prototype #1 by adding significant functionality:

Feature	Prototype #1	Prototype #2	Implementation Details
Core spam detection	**+**	**+**	BERT-based binary classification
Frontend-backend connection	**+**	**+**	Flask Blueprint with Jinja2 templates
Keyword explanations	**-**	**+**	IntegratedGradients
Error handling with retry	**-**	**+**	Exponential backoff
Comprehensive logging	**-**	**+**	File rotation, error separation, console output
Resource preloading	**-**	**+**	HTTP Link headers for CSS and fonts
Singleton pattern	**-**	**+**	SpamService caches SpamChecker instance
Alternative classifier	**-**	**+**	GeneralRiskOverseer with sliding windows

5.2 New Components in Prototype #2

1. Keyword Explanation System (`spam_checker.py`)

```
def explain(self, text: str, top_k: int = 5) -> list[tuple[str, float]]:
    from captum.attr import IntegratedGradients

    # Create baseline (all PAD tokens)
    baseline_ids = torch.full_like(input_ids, fill_value=self.tokenizer.pad_token_id)

    # Calculate attributions
    ig = IntegratedGradients(forward_with_embeds)
    attributions = ig.attribute(
        inputs=input_embeds,
        baselines=baseline_embeds,
        additional_forward_args=(attention_mask,),
        n_steps=24,  # 24 interpolation steps
    )

    # Merge wordpieces and filter
    token_scores = attributions.sum(dim=-1).squeeze(0).detach().cpu().numpy().tolist()
    merged = _merge_wordpieces(tokens, token_scores)
    return merged[:top_k]
```

2. Retry Logic with Exponential Backoff (`routes_spam.py`)

```
max_retries = 3
retry_delay = 1
for attempt in range(max_retries):
    try:
        prediction = SpamService.classify_text(text, explain=True, top_k=5)
        return render_template("check_spam.html", show_result=True, ...)
    except (ConnectionError, ConnectionResetError, Timeout, RuntimeError) as e:
        if attempt == max_retries - 1:
            logger.error(f"Max retries exceeded after {max_retries} attempts")
            return render_template("404.html")
        wait_time = retry_delay * (2 ** attempt)
        time.sleep(wait_time)
```

3. Professional Logging System (`logger.py`)

```
class Logger:
    _loggers: dict[str, logging.Logger] = {}

    @classmethod
    def get_logger(cls, name: str) -> logging.Logger:
        if name in cls._loggers:
            return cls._loggers[name]
        logger = logging.getLogger(name)
        logger.setLevel(LOG_LEVEL)
        cls._setup_handlers(logger)  # File, Error, Console handlers
        cls._loggers[name] = logger
        return logger
```

4. HTTP Resource Preloading (`routes_spam.py`)

```
@spam_bp.after_request
def push_resources(response):
    """Add preload headers for static resources."""
    if response.status_code == 200 and 'text/html' in response.content_type:
        response.headers.add(
            'Link',
            '</static/css/styles.css>; rel=preload;as=style')
        response.headers.add(
            'Link',
            '<https://fonts.googleapis.com/icon?family=Material+Icons>; rel=preload;as=style')
    return response
```

5.3 Performance Improvements

Metric	Prototype #1	Prototype #2	Improvement
Model load time	~3s per request	~3s first request only	Singleton caching
Classification time	~200ms	~200ms	Same (optimized)
With explanation	NAN	~0.5-3s	New feature

Metric	Prototype #1	Prototype #2	Improvement
Transient error recovery	NAN	NAN	Retry mechanism
Resource load time	~500ms	~300ms	HTTP preloading
Debugging capability	Medium	High	Logging

6. Conclusion

6.1 Key Design Decisions

1. Production-Ready Error Handling:
    - **Retry logic**: Up to 3 attempts with exponential backoff
    - **Exception granularity**: Different handling for ConnectionError, Timeout, RuntimeError
    - **Comprehensive logging**: File rotation, error separation, console output
    - **User-friendly errors**: Generic 404 page, no technical details exposed
2. Explainable AI:
    - **IntegratedGradients**: Industry standard attribution method
    - **Wordpiece handling**: Proper BERT subword token merging
    - **Top-K selection**: Display 5 most influential keywords
3. Performance Optimization:
    - **Singleton pattern**: Model loaded once, reused across requests
    - **HTTP preloading**: CSS and fonts preloaded via Link headers
    - **Lazy initialization**: Model loaded only when first needed
    - **Batch processing**: GeneralRiskOverseer processes windows in parallel

6.2 Prototype #2 Features Summary

Feature	Implementation	Code Location	Complexity
Spam Detection	BERT binary classification	`spam_checker.py:predict()`	Medium
Explanations	IntegratedGradients	`spam_checker.py:explain()`	High

Feature	Implementation	Code Location	Complexity
	attribution		
Retry Logic	Exponential backoff (1s, 2s, 4s)	`routes_spam.py:show_form()`	Medium
Logging	3 handlers (file, error, console)	`logger.py:Logger`	Low
Resource Preload	HTTP Link headers	`routes_spam.py:push_resources()`	Low
Singleton	Lazy-loaded SpamChecker cache	`spam_checker.py:SpamService`	Low
Alternative Classifier	Sliding window processing	`gro.py:GeneralRiskOverseer`	Medium

6.3 Final Remarks

The behavioral modeling for Prototype #2 demonstrates a mature, production-ready architecture that:

- **Respects user privacy** through complete statelessness
- **Provides transparency** through detailed interaction diagrams and explainable AI
- **Maintains simplicity** with clear state machines and activity flows
- **Ensures resilience** through comprehensive error handling
- **Delivers performance** through singleton pattern and optimization

The sequence diagrams, state machines, and activity diagrams provide a comprehensive blueprint for implementation and serve as reference documentation for maintenance and future extensions. All diagrams are based directly on the actual code implementation, ensuring accuracy and maintainability.