

# **GSOC 2025 - Proposal for RSPAMD**

## **Multi-Class Bayesian Classifier**

### **Mentors:**

Vsevolod Stakhov([vsevolod@rspamd.com](mailto:vsevolod@rspamd.com))

Andrew Lewis([alewis@rspamd.com](mailto:alewis@rspamd.com))

# Table of Contents-

1. [Personal Information](#)
2. [Project Overview](#)
  - 2.1 [Project Synopsis](#)
  - 2.2 [Deliverables](#)
3. [Approach & Implementation Plan](#)
  - 3.1 [Bayesian Classifier Enhancement](#)
    - 3.1.1 [Understanding the Bayesian Model in Rspamd](#)
    - 3.1.2 [Implementing Multi-Class Classification in lua scripts](#)
  - 3.2 [GPT Plugin Integration](#)
    - 3.2.1 [Why Integrate GPT?](#)
    - 3.2.2 [Exploring Other LLM Models for Enhanced Classification](#)
    - 3.2.3 [Performance Benchmarking & Model Evaluation](#)
    - 3.2.4 [Hybrid Approach: Bayesian + LLM Ensemble](#)
    - 3.2.5 [Challenges & Solutions](#)
  - 3.3 [Training and Performance Benchmarking](#)
    - 3.3.1 [Dataset Preparation](#)
    - 3.3.2 [Performance Evaluation](#)
  - 3.4 [Communication with Mentors](#)
  - 3.5 [Drive link for updated scripts](#)
  - 3.6 [Forked copy of Rspamd](#)
4. [Step-by-step Timeline](#)
5. [Candidate Details](#)
  - 5.1 [Why Am I Motivated by this project?](#)
  - 5.2 [Why Am I a Good Candidate?](#)
  - 5.3 [Past Experience](#)
  - 5.4 [References](#)
  - 5.5 [Availability](#)
6. [Conclusion](#)

# 1. Personal Information

- **Name:** Kevin Shah
- **Email:** [kevinshah3008@gmail.com](mailto:kevinshah3008@gmail.com)
- **GitHub:** <https://github.com/kevinzb56>
- **Location:** Mumbai, India
- **University:** Veermata Jijabai Technological Institute
- **Degree:** B.Tech in Computer Engineering
- **Resume:**  resume\_kevin.pdf
- **Timezone:** IST (UTC +5:30)
- **Telephone:** +91 9930022746

## 2. Project Overview

### 2.1 Project Synopsis

In today's fast-paced world, we receive an overwhelming amount of emails daily. As a result, there is a growing need for an efficient mechanism to classify these emails into different categories for better organization and prioritization. While Rspamd's Bayesian classifier currently supports only binary classification (spam/ham), this project aims to extend its capabilities. The goal is to enhance the current Bayesian classifier to support multiple configurable categories, such as important, finance, personal, promotional, spam, and social, to streamline email management. Additionally, the project will integrate an AI-driven learning mechanism using a GPT plugin to dynamically update classification models based on evolving email patterns.

### 2.2 Deliverables

- Extend Rspamd's Bayesian classifier for multi-class classification.
- Implement AI-driven learning via a GPT plugin for continuous model updates.

## 3. Approach & Implementation Plan

### 3.1 Bayesian Classifier Enhancement

To extend Rspamd's Bayesian classifier to support multi-class classification, first I would:

1. Understand the current Bayesian classifier implementation (how ham/spam is classified).
2. Modify it to handle multiple categories (e.g., **Finance, Personal, Promotional, Spam, Important, Social**).

These categories would be configurable by the user and which categories would be implemented can be further discussed with the mentors. For a general assumption, I am assuming the following 6 categories, namely **Finance, Personal, Promotional, Spam, Important, Social**.

#### 3.1.1 Understanding the Bayesian Model in Rspamd

**Major** changes are required in the following files:

##### 1. **bayes.c**

The core bayesian implementation lies in the **libstat** directory. And in the classifiers we first have **bayes.c** which implements a **Bayesian classifier** for spam detection. It calculates the probability of spam/ham using **Bayes' theorem** to assign the category. The classifier assigns probabilities to words in emails, maintains a statistical model, and updates based on learned data. In summary, the following steps are followed

1. **Training:**
  - Extract words from an email.
  - Mark them as spam or ham.
  - Store/update probabilities in the model.
2. **Classification:**
  - Extract words from a new email.
  - Compute spam probability for each word.
  - Combine probabilities using Bayes' theorem.
  - Compare final probability to thresholds.
  - Assign classification (spam, ham, or unknown).
3. **Updating:**
  - Continuously improve accuracy by learning from new emails.

To extend to multi-class classification, we can follow these steps in **bayes.c**:

##### 1. Modify Data Structures to Support Multiple Classes

```
#define NUM_CLASSES 6
```

```

typedef enum {
    CLASS_SPAM = 0,
    CLASS_SOCIAL,
    CLASS_FINANCE,
    CLASS_PERSONAL,
    CLASS_PROMOTIONAL,
    CLASS_IMPORTANT,
} bayes_class_t;

static const char *class_names[NUM_CLASSES] = {
    "spam", "social", "finance", "personal", "promotional", "important"
};

struct bayes_task_closure {
    double class_prob[NUM_CLASSES]; // Store probability for each category
    uint64_t processed_tokens;
    uint64_t total_hits;
    uint64_t text_tokens;
    struct rspamd_task *task;
};

```

## 2. Modify Token Classification to Support Multi-Class

Each token should now be classified across multiple categories. **Now calculate probability for all classes instead of just spam/ham.**

```

static void bayes_classify_token(struct rspamd_classifier *ctx,
                                rspamd_token_t *tok, struct
                                bayes_task_closure *cl) {
    unsigned int i;
    int id;
    unsigned int class_counts[NUM_CLASSES] = {0};
    unsigned int total_count = 0;
    struct rspamd_statfile *st;
    struct rspamd_task *task;
    double class_freq[NUM_CLASSES], bayes_class_prob[NUM_CLASSES], fw,
    w, val;

    task = cl->task;

    /* Iterate through all statfiles (classes) */
    for (i = 0; i < ctx->statfiles_ids->len; i++) {
        id = g_array_index(ctx->statfiles_ids, int, i);
        st = g_ptr_array_index(ctx->statfiles, id);
        g_assert(st != NULL);
    }
}

```

```

    val = tok->values[id];

    if (val > 0) {
        class_counts[id] += val;
        total_count += val;
        cl->total_hits += val;
    }
}

/* Proceed only if the token appears frequently enough */
if (total_count >= ctx->cfg->min_token_hits) {

    /* Calculate frequency for each class */
    for (i = 0; i < NUM_CLASSES; i++) {
        class_freq[i] = ((double)class_counts[i] / MAX(1.,
(double)ctx->spam_learns));
    }

    /* Feature weight calculation (extensible for token type) */
    switch (tok->flags & RSPAMD_STAT_TOKEN_MASK) {
        case RSPAMD_STAT_TOKEN_FLAG_UNIGRAM:
            fw = 1.0;
            break;
        case RSPAMD_STAT_TOKEN_FLAG_BIGRAM:
            fw = 1.2; // example: give higher weight to bigrams
            break;
        case RSPAMD_STAT_TOKEN_FLAG_TRIGRAM:
            fw = 1.5; // example: even more for trigrams
            break;
        default:
            fw = 1.0;
            break;
    }

    /* Calculate Bayesian probability for each class */
    w = (fw * total_count) / (1.0 + fw * total_count);

    for (i = 0; i < NUM_CLASSES; i++) {
        bayes_class_prob[i] = ((w * 0.5) + (total_count *
class_freq[i])) / (w + total_count);

        /* Accumulate log-probability for each class */
        cl->class_prob[i] += log(bayes_class_prob[i]);
    }
}

```

```

        cl->processed_tokens++;

        if (!(tok->flags & RSPAMD_STAT_TOKEN_FLAG_META)) {
            cl->text_tokens++;
        }
    }
}

```

- **Uses log probabilities** to avoid floating-point precision issues.

### 3. Modify Classification Function

The classifier should now assign an email to **one of multiple classes** instead of just spam/ham.

```

gboolean bayes_classify(struct rspamd_classifier *ctx,
                      GPtrArray *tokens, struct rspamd_task *task) {
    struct bayes_task_closure cl;
    rspamd_token_t *tok;
    unsigned int i, text_tokens = 0;
    double max_prob = -1.0;
    bayes_class_t best_class = CLASS_SPAM; // Default to spam

    g_assert(ctx != NULL);
    g_assert(tokens != NULL);

    memset(&cl, 0, sizeof(cl));
    cl.task = task;

    for (i = 0; i < tokens->len; i++) {
        tok = g_ptr_array_index(tokens, i);
        if (!(tok->flags & RSPAMD_STAT_TOKEN_FLAG_META)) {
            text_tokens++;
        }
    }

    if (text_tokens == 0) {
        msg_info_task("skipped classification as there are no text
tokens.");
        return TRUE;
    }

    for (i = 0; i < tokens->len; i++) {
        tok = g_ptr_array_index(tokens, i);
        bayes_classify_token(ctx, tok, &cl);
    }
}

```

```

    if (cl.processed_tokens == 0) {
        msg_info_bayes("No tokens found in bayes database, ignoring
classification.");
        return TRUE;
    }

    /* Find the most likely class */
    for (i = 0; i < NUM_CLASSES; i++) {
        if (cl.class_prob[i] > max_prob) {
            max_prob = cl.class_prob[i];
            best_class = i;
        }
    }

    /* Assign classification result */
    msg_info_task("Email classified as: %s (probability: %.3f)",
class_names[best_class], max_prob);

    return TRUE;
}

```

- Finds the class with the **highest probability** so as to assign that class to the mail.

#### 4. Modify Training Function to Support Multi-Class

Modified Training Logic for Multi-Class:

```

void bayes_learn(struct rspamd_classifier *ctx,
struct rspamd_task *task, gboolean is_spam) {
    bayes_class_t assigned_class = determine_class_from_labels(task);
    struct rspamd_statfile *st;
    rspamd_token_t *tok;
    unsigned int i;
    for (i = 0; i < ctx->statfiles_ids->len; i++) {
        int id = g_array_index(ctx->statfiles_ids, int, i);
        st = g_ptr_array_index(ctx->ctx->statfiles, id);
        g_assert(st != NULL);
        /* Adjust token probabilities for assigned class */
        tok->values[assigned_class] += 1.0;
    }
    msg_info_task("Learning: Assigned %s to email",
class_names[assigned_class]);
}

```



- Determines **which class the email belongs to** before updating the model.
- Adjusts token probabilities accordingly.

## 2. classifier.h

Next we can make following changes in **classifier.h**:

We can **update bayes\_classify** method to handle multi-class classification:

Added `bayes_class_t *result_class` to return the classified category.

```
gboolean bayes_classify(struct rspamd_classifier *ctx,
                      GPtrArray *tokens,
                      struct rspamd_task *task,
                      bayes_class_t *result_class);
```

## Update Learning Function(bayes\_learn)

```
gboolean bayes_learn(struct rspamd_classifier *ctx,
                    GPtrArray *tokens,
                    struct rspamd_task *task,
                    bayes_class_t assigned_class,
                    gboolean unlearn,
                    GError **err);
```

Instead of `is_spam` in the current implementation, it now takes `bayes_class_t assigned_class`.

## 3. lua\_classifier.c

Update `lua_classifier_classify` method to return multi-class labels:

```
gboolean
lua_classifier_classify(struct rspamd_classifier *cl,
                      GPtrArray *tokens,
                      struct rspamd_task *task)
{
    struct rspamd_lua_classifier_ctx *ctx;
    struct rspamd_task **ptask;
    struct rspamd_classifier_config **pcfg;
    lua_State *L;
    rspamd_token_t *tok;
```

```

unsigned int i;
uint64_t v;

ctx = g_hash_table_lookup(lua_classifiers, cl->subrs->name);
g_assert(ctx != NULL);
L = task->cfg->lua_state;

lua_rawgeti(L, LUA_REGISTRYINDEX, ctx->classify_ref);
ptask = lua_newuserdata(L, sizeof(*ptask));
*ptask = task;
rspamd_lua_setclass(L, rspamd_task_classname, -1);
pcfg = lua_newuserdata(L, sizeof(*pcfg));
*pcfg = cl->cfg;
rspamd_lua_setclass(L, rspamd_classifier_classname, -1);

lua_createtable(L, tokens->len, 0);
for (i = 0; i < tokens->len; i++) {
    tok = g_ptr_array_index(tokens, i);
    v = tok->data;
    lua_createtable(L, 4, 0);
    lua_pushinteger(L, (uint32_t) (v >> 32));
    lua_rawseti(L, -2, 1);
    lua_pushinteger(L, (uint32_t) (v));
    lua_rawseti(L, -2, 2);
    lua_pushinteger(L, tok->window_idx);
    lua_rawseti(L, -2, 3);
    lua_pushinteger(L, tok->category); // Multi-class category
    lua_rawseti(L, -2, 4);
    lua_rawseti(L, -2, i + 1);
}

if (lua_pcall(L, 3, 0, 0) != 0) {
    msg_err_luacl("error running classify function for %s: %s",
ctx->name,
lua_tostring(L, -1));
    lua_pop(L, 1);
    return FALSE;
}

return TRUE;
}

```

Update lua\_classifier\_learn method to learn from multiple categories:

It updates learning logic to store probabilities for multiple categories.

```

gboolean
lua_classifier_learn(struct rspamd_classifier *cl,

```

```

        GPtrArray *tokens,
        struct rspamd_task *task,
        int category, // New parameter for multi-class
        gboolean unlearn,
        GError **err)
{
    struct rspamd_lua_classifier_ctx *ctx;
    struct rspamd_task **ptask;
    struct rspamd_classifier_config **pcfg;
    lua_State *L;
    rspamd_token_t *tok;
    unsigned int i;
    uint64_t v;

    ctx = g_hash_table_lookup(lua_classifiers, cl->subrs->name);
    g_assert(ctx != NULL);
    L = task->cfg->lua_state;

    lua_rawgeti(L, LUA_REGISTRYINDEX, ctx->learn_ref);
    ptask = lua_newuserdata(L, sizeof(*ptask));
    *ptask = task;
    rspamd_lua_setclass(L, rspamd_task_classname, -1);
    pcfg = lua_newuserdata(L, sizeof(*pcfg));
    *pcfg = cl->cfg;
    rspamd_lua_setclass(L, rspamd_classifier_classname, -1);

    lua_createtable(L, tokens->len, 0);
    for (i = 0; i < tokens->len; i++) {
        tok = g_ptr_array_index(tokens, i);
        v = tok->data;
        lua_createtable(L, 4, 0);
        lua_pushinteger(L, (uint32_t) (v >> 32));
        lua_rawseti(L, -2, 1);
        lua_pushinteger(L, (uint32_t) (v));
        lua_rawseti(L, -2, 2);
        lua_pushinteger(L, tok->window_idx);
        lua_rawseti(L, -2, 3);
        lua_pushinteger(L, category); // Multi-class category
        lua_rawseti(L, -2, 4);
        lua_rawseti(L, -2, i + 1);
    }

    lua_pushboolean(L, unlearn);
    lua_pushinteger(L, category);

    return lua_pcall(L, 5, 0, 0) == 0;
}

```

#### 4. hyperscan\_tools.cxx

hyperscan\_tools.cxx is yet another important file which needs to be updated and we'll need to adjust the handling of pattern matching and classification. Some possible modifications that can be made to these functions:

Function name	Modification
rspamd_hyperscan_maybe_load	Loads patterns with <b>multi-class labels</b> .
convert_to_multiclass()	Ensures each pattern has <b>only one category</b> .
select_best_category()	Resolves conflicts when multiple categories match.
rspamd_hyperscan_get_category	Returns <b>one</b> category per email.
rspamd_hyperscan_from_raw_db	Supports multi-class classification from raw DB files.

#### 5. expression.c

We have **expression.c** which is responsible for parsing and evaluating expressions using a syntax tree and stack-based approach

1. Extend rspamd\_expression\_op:
  - Need new operators or handling for multi-class scoring, e.g., CATEGORY\_A, CATEGORY\_B, etc.
2. Modify rspamd\_expression\_elt to Store Multiple Class Scores
  - Multi-class classification needs an array or hash table to store probability scores for multiple categories.
3. Adjust rspamd\_expr\_stack\_push() and rspamd\_expr\_stack\_pop()
  - These functions need to handle multi-class elements instead of single values.
4. Update Logical and Comparison Operators for Multi-Class
  - We can consider using softmax probabilities instead of strict boolean evaluations.

More details about the code in the [drive](#)

## 6. shingles(lua\_shingles.cxx, shingles.h)

**Shingles** are another aspect that we have to consider.

### Multi-Class Support added:

- Each shingle is linked to a category (sh->categories[i]).
- New function get\_class\_hashes(category) retrieves hashes per category.
- Function get\_all\_classes() returns all available categories.

### Improved Usability:

- Uses std::set to ensure unique category retrieval.
- Efficiently maps shingles to their respective classes.

## 3.1.2 Implementing Multi-Class Classification in lua scripts

**Major** changes are required in the following files:

### 1. rspamd.lua

rspamd/rules/rspamd.lua is the **main** config file in rspamd as it loads various Lua scripts from RULESDIR (e.g., archives.lua, headers.lua) that define symbols and rules for spam detection.

Approach: Add category-specific symbols by introducing prefixes (e.g., FINANCE\_, PROMOTIONAL\_, etc) to symbol names in the rules.

Changes:

1. **Rule Files:** Modify the included .lua files (e.g., headers\_checks.lua) to define category-prefixed symbols.
2. **rspamd.lua:** Add logic to register or validate these categorized symbols globally, ensuring consistency.

Since rspamd.lua is the starting point, we should add a table of categories and a function to register symbols with category prefixes.

```

local categories = {
  "FINANCE", "IMPORTANT", "PERSONAL", "PROMOTIONAL", "SOCIAL", "SPAM"
}

-- Function to register a symbol with a category prefix
local function register_categorized_symbol(category, name, opts)
  if not categories[category] then
    local rspamd_logger = require "rspamd_logger"
    rspamd_logger.errx(rspamd_config, "Invalid category: %s for symbol %s",
category, name)
    return
  end
  local full_name = category .. "_" .. name
  opts.name = full_name
  rspamd_config:register_symbol(opts)
end

```

Changes in Rule Files: Update symbol definitions to include category prefixes (e.g., FINANCE\_FORGED\_SENDER).

## 2. example(in headers\_checks.lua):

```

register_categorized_symbol('FINANCE', 'FORGED_SENDER', {
  score = 0.3,
  description = 'Sender is forged in a finance-related email',
  group = 'headers'
})

register_categorized_symbol('PROMOTIONAL', 'FORGED_SENDER', {
  score = 0.5,
  description = 'Sender is forged in a promotional email',
  group = 'headers'
})

```

Symbols like FINANCE\_FORGED\_SENDER and PROMOTIONAL\_FORGED\_SENDER will appear in the config dump (via configdump.c), enabling downstream classification logic to distinguish email categories.

The updated configdump.c will parse these prefixes and output a category field, making the multiclass structure visible to users and tools.

## 3. Category detection logic in configdump.c:

```

const char *category = "unknown";

```

```

if (strncmp(sym_name, "FINANCE_", 8) == 0) {
    category = "finance";
} else if (strncmp(sym_name, "IMPORTANT_", 10) == 0) {
    category = "important";
} else if (strncmp(sym_name, "SPAML_", 9) == 0) {
    category = "spam";
} /* ... and so on ... */

```

If no prefix matches, it defaults to "unknown". (We can have a better fallback mechanism like defaulting to personal category)

**Before:** If we run `rspamadm configdump -d` (for an example)

```

kevin@DESKTOP-219SJL0:~$ sudo rspamadm configdump -d
[sudo] password for kevin:
symbols {
    APPLE_IOS_MAILER {
        score = 0.0;
        description = "Sent with Apple iPhone/iPad Mail";
        type = "filter";
        disabled = false;
        one_shot = true;
        group = "headers";
    }
}

```

**After:**

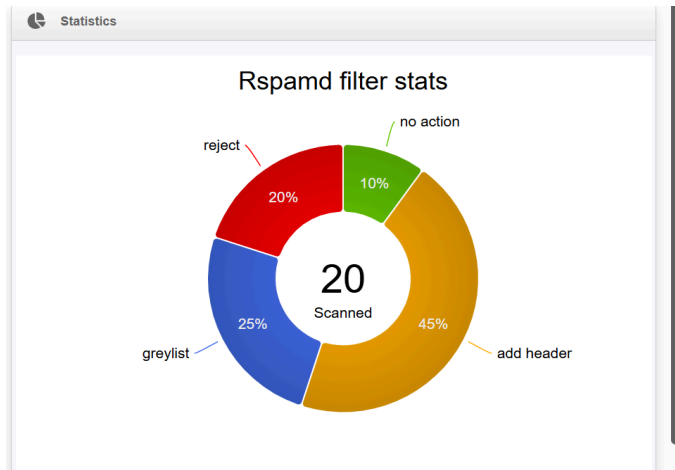
```

{
  "symbols": {
    "FINANCE_FORGED_SENDER": {
      "score": 0.3,
      "description": "Sender is forged in a finance email...",
      "disabled": false,
      "one_shot": false,
      "category": "finance"
    }
  }
}

```

The prefix in the symbol name (FINANCE\_) is parsed, and the category field explicitly shows "finance."

#### 4. force\_actions.lua:



- Currently, the script enforces single-category actions (e.g., reject, no action, add header).
- Modify it to **support multiple classifications** (e.g., spam, important, finance, personal, etc).
- Extend lua\_selectors to extract more metadata (e.g., email length, attachment type).

Other extended lua files in the [drive](#)

## 3.2 GPT Plugin Integration

### 3.2.1 Why Integrate GPT?

Rspamd's Bayesian classifier is effective at learning from given email data, but it has limitations:

- It requires **manual training** and periodic updates.
- It struggles with **contextual understanding**, and hence it can't get the difference between the same word but applied in different situations. For example, we have two sentences where "bank" is used in different contexts:

**River Bank Context:** *The children sat by the river bank, skipping stones across the calm water.*

**Money Bank Context:** *She went to the bank to deposit her paycheck and withdraw some cash.*

- It does not **adapt dynamically** to emerging threats or new email patterns.



To address these limitations, integrating Large Language Models (LLMs) such as GPT into Rspamd offers a dynamic and adaptive approach to spam classification. GPT's deep contextual understanding allows it to identify complex patterns that traditional classifiers might overlook. By leveraging GPT's strengths, Rspamd can enhance its filtering capabilities in multiple ways:

- **Context-Aware Classification:** Unlike Bayes, which relies on static probability distributions, GPT can discern meaning based on sentence structure, intent, and semantics.
- **Adaptive Learning:** GPT can continuously refine Rspamd's classification accuracy by supplementing the Bayesian model with real-time contextual insights.
- **Reduced Training Overhead:** Instead of relying solely on manually labeled datasets, GPT can provide heuristic training signals, reducing the need for human intervention.

By integrating AI-driven learning via a GPT plugin in Rspamd, we aim to enhance the existing Bayesian filtering system with dynamic model updates for the additional classes we have introduced. This hybrid approach not only improves classification accuracy but also ensures adaptive learning, reducing false positives and strengthening Rspamd's ability to detect evolving email threats in real time.

How does **gpt.lua** work?

1. It sends email content (parts) to an LLM API for classification.
2. The LLM's response is parsed to extract category labels.
3. Results are stored in a cache (rspamd\_map\_add), preventing redundant API calls.
4. It compares the LLM result against a predefined consensus threshold before applying classification.

### Modify LLM Prompting & Parsing

- Instead of a strict binary response, request the LLM to output a category label.
- Example:

```
local llm_query = "Classify this email into one of these categories: spam, important, finance, personal, promotional, social. Return only the category name."
```

```
local category = string.match(out, "(%w+)") -- Extracts the first word (assuming it's the category)
```

### 3.2.2 Exploring Other LLM Models for Enhanced Classification

While GPT-4o provides strong classification capabilities, its reliance on API calls raises concerns about cost, latency, and data privacy. To further optimize classification, we explore alternative LLM models that can be fine-tuned or hosted locally:

#### Self-Hosted Models for Cost-Effective Processing

- **LLaMA 3** – A lightweight alternative that can be fine-tuned for spam detection and deployed on-premises.
- **Mistral-7B** – Offers efficiency comparable to GPT-4 with lower computational costs.
- **Falcon-40B** – An open-weight model capable of handling nuanced text classification.

By running a self-hosted model, we mitigate dependency on third-party APIs and improve inference speeds.

#### Fine-Tuning a Custom BERT Model

To further enhance accuracy, we can fine-tune **BERT (or RoBERTa)** on a dedicated email classification dataset. Steps:

1. **Dataset Collection** – Curate a **larger** dataset with emails labeled as **spam, finance, personal, social, promotional, and important**.
2. **Preprocessing** – Tokenize emails, remove unnecessary metadata, and clean text.
3. **Fine-Tuning** – Train BERT/RoBERTa with supervised learning on the curated dataset.
4. **Integration with Rspamd** – Deploy a fine-tuned model within Rspamd's pipeline for classification.

Fine-tuning allows for a **task-specific model** that learns domain-specific patterns in email classification.

For example, BERT performs quite well on these toy examples that I've tested on and it classifies the emails pretty well.

```
# Testing on toy examples
subjects = ["Free gift cards", "Hello Dear", "Your account has been compromised.",
            "Thank you for Participating!", "You have 23 notifications about Vidhi and others",
            "Exclusive Discount Just for You!", "Congratulations! You've won a lottery!",
            "Urgent: Team Meeting Tomorrow"]
texts = ["You have won a free gift card. Click here to claim!",
         "I am stuck in Africa and I need your help.",
         "Kindly login and reclaim your account.",
         "Thank you for your participation in Goldman Sachs Hackathon event! We were impressed with the creati",
         "You have 23 unread notifications to review. A lot has happened on Facebook since you last logged in.",
         "Get 20% off your next purchase! Use code: SAVE20.",
         "Click here to claim your prize of $1,000,000!",
         "Dear Team, please be informed that we have an urgent meeting scheduled for tomorrow at 10 AM to disc"]
predicted_classes = [classify_email(subject, text).item() for subject, text in zip(subjects, texts)]
print(predicted_classes)

[5, 2, 0, 4, 1, 3, 5, 1]
```

We can expect an output like this from the above script, where the different classes are listed below.

classes:

"""

0 - finance  
 1 - important  
 2 - personal  
 3 - promotional  
 4 - social  
 5 - spam  
 """

### 3.2.3 Performance Benchmarking & Model Evaluation

To validate the effectiveness of LLM integration, we conduct **comparative testing** between the baseline Bayesian classifier, GPT-4o, and other explored models.

#### Test Methodology:

1. **Dataset** – A balanced dataset of emails from real-world sources.
2. **Metrics** – Compare models based on:
  - **Accuracy** =  $(TP + TN) / (\text{Total samples})$
  - **Precision** =  $TP / (TP + FP)$
  - **Recall** =  $TP / (TP + FN)$
  - **F1 Score** =  $2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

- **Inference Time** – Time taken for classification per email.
- **Cost Analysis** – API vs. self-hosted models.

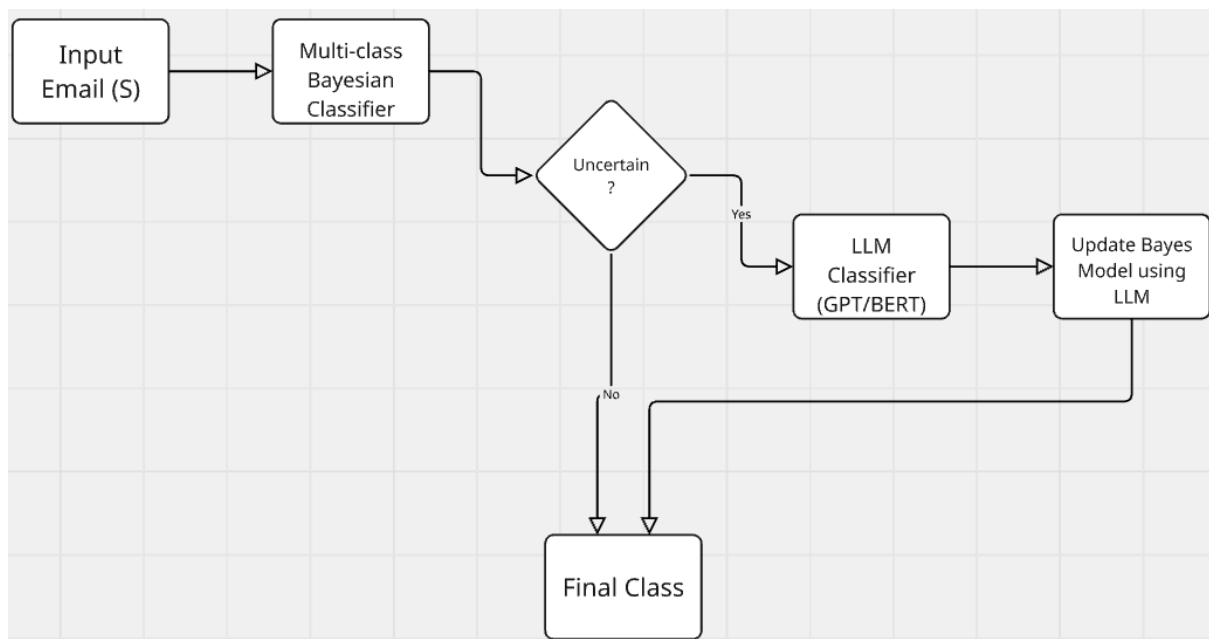
### 3.2.4 Hybrid Approach: Bayesian + LLM Ensemble

I propose a **grouped method** where:

1. **Rspamd first applies Bayesian filtering.**
2. **If uncertainty > threshold, LLM is used for re-evaluation.**
3. **GPT/BERT assigns a confidence score, refining classification.**
4. **Rspamd updates the Bayesian classifier dynamically** using LLM predictions.

This hybrid model ensures **fast, accurate, and cost-effective filtering** while improving dynamically over time. However this approach can be further discussed with the mentors and modified if needed.

Overall skeleton looks like this:



This diagram represents a **multi-class classification pipeline** using a **Multinomial Bayes' Model**. If the classifier is uncertain about the email's category, it is passed to a **BERT/GPT model** for accurate classification. The correct label is then stored, improving future predictions.

### 3.2.5 Challenges & Solutions

## 1. Limited Dataset for Multi-Class Training (Primary Challenge)

- One major limitation is the **lack of a large, labeled dataset** with multiple email categories. To overcome this, we:
  - Can use an **Auto-labeller** architecture which will automatically label the emails of our dataset(Have implemented this and it proves to be a pretty accurate solution to this problem)
  - **Augment data** by synthesizing labeled emails based on existing datasets.
  - **Leverage transfer learning**, fine-tuning **BERT** on publicly available email datasets to improve accuracy.

## 2. Latency Issues in API Calls

- **API requests to GPT introduce delays**, so we will optimize the classification pipeline:
  - **Threshold-Based Invocation:** Rspamd will classify emails using **Bayesian filtering** first.
  - **Fast Local Processing:** If uncertain, it will pass the email to **fine-tuned BERT** for a quick decision.

## 3. Self-Learning & Continuous Improvement

- To improve over time, we will **dynamically update the Bayesian model**:
  - **Feedback Loop:** Responses from **BERT** and **GPT** will be stored and **used to retrain the model**.
  - **Adaptive Classification:** As the dataset grows, the system will rely more on fine-tuned BERT and **less on costly API calls**.
  - **Regular Performance Tuning:** Testing classification accuracy, refining thresholds, and **optimizing response times** for real-world deployment.

## 4. UI integration

- Update the UI to accommodate additional classes by introducing a more detailed, interactive chart (e.g., a stacked bar chart or an expandable donut chart) that can dynamically display new categories.
- **Scalable Design Framework:** Implement a responsive and modular UI framework that can adapt to an increasing number of classes over time.

## 3.3 Training and Performance Benchmarking

### 3.3.1 Dataset Preparation

- Collect real-world email samples covering multiple categories.
- Balance the dataset to avoid class bias.

### 3.3.2 Performance Evaluation

- Compare Bayesian-only vs. GPT-enhanced classification.
- Evaluate using Precision, Recall, and F1-score.


```
1 from sklearn.metrics import classification_report
2
3 def evaluate_model(predictions, ground_truth):
4     report = classification_report(ground_truth, predictions)
5     print(report)
```

### 3.4 Communication with Mentors

1. I will maintain regular communication with my mentors, providing consistent updates on my progress and any challenges I face.
2. I will use their preferred communication platform (email, Telegram, or any other as per their convenience) to ensure smooth interactions.
3. I plan to have weekly meetings with my mentors to discuss my approach, receive feedback, and implement suggested improvements.
4. In case of urgent issues, I will promptly reach out for clarifications or guidance to keep the project on track.

### 3.5 Drive link for updated scripts

Drive link with all the updated files that could be implemented for multi-class classifier:

 Rspamd updated scripts

### 3.6 Forked copy of Rspamd

Forked copy of Rspamd where I've started integrating multi-class bayesian classifier on current implementation:

<https://github.com/kevinzb56/rspamd/tree/kevin>

## 4. Step-by-Step Timeline (22 Weeks)

Date and Time	Event	Proposed Tasks
Pre-GSoC (Before May 8)	Preparation Phase	<ul style="list-style-type: none"><li>- Understand rspamd's codebase and explore documentation to get deeper insights of current implementation.</li><li>- Research Bayesian classifiers, Rspamd architecture, and existing Bayes module.</li><li>- Learn Lua, including its scripting styles and best practices, as required for implementation.</li></ul>
May 8 - June 1	Community Bonding Period	<ul style="list-style-type: none"><li>- Engage with mentors and community.</li><li>- Define category thresholds and probability distributions.</li><li>- Start working on dataset generation.</li><li>- Finalize dataset generation strategies.</li></ul>
June 2 - June 8 <b>Week 1</b>	Coding Begins	<ul style="list-style-type: none"><li>- Set up the development environment locally.</li><li>- Implement initial modifications in the existing Bayes modules for multi-class support.</li><li>-Refine dataset and begin preprocessing.</li></ul>
June 9 - June 15 <b>Week 2</b>	Extend Bayes Module for Multi-Class Support	<ul style="list-style-type: none"><li>- Modify probability distributions to support multi-class classification.</li><li>- Implement token weighting and probability updates for each category.</li></ul>

		<ul style="list-style-type: none"> <li>- Update the Redis-backed storage format to accommodate multiple categories.</li> </ul>
June 16 - June 22 <b>Week 3</b>	Training & Testing Multi-Class Classifier	<ul style="list-style-type: none"> <li>- Train the Bayesian model with real-world email data.</li> <li>- Start writing test cases for multi-class predictions.</li> </ul>
June 23 - June 29 <b>Week 4</b>	Debugging & Optimization	<ul style="list-style-type: none"> <li>- Fix issues related to overlapping classifications and incorrect probability updates.</li> <li>- Optimize performance of multi-class probability calculations.</li> </ul>
June 30 - July 6 <b>Week 5</b>	GPT Plugin Prototype Development	<ul style="list-style-type: none"> <li>- Refine developed Lua-based GPT plugin (gpt.lua) for Rspamd.</li> <li>- Implement a function to send email content to GPT via an API.</li> <li>- Introduce basic caching (rspamd_map_add) to store classification results and reduce redundant API calls.</li> </ul>
July 7 - July 13 <b>Week 6</b>	Midterm Evaluation Preparation	<ul style="list-style-type: none"> <li>- Ensure basic multi-class support is working.</li> <li>- Validate and test the GPT plugin prototype.</li> <li>- Prepare the midterm evaluation report with test results</li> </ul>
July 14 - July 18 <b>Week 7</b>	Midterm Evaluation	<ul style="list-style-type: none"> <li>- Submit midterm evaluation.</li> <li>- Receive feedback from mentors.</li> </ul>
July 19 - July 25 <b>Week 8</b>	Buffer Time	<ul style="list-style-type: none"> <li>- Complete any missed work, light documentation, and minor fixes.</li> </ul>



		<ul style="list-style-type: none"> <li>- Improve dataset quality if needed before GPT integration.</li> </ul>
July 26 - August 1 <b>Week 9</b>	Enhancing GPT Plugin	<ul style="list-style-type: none"> <li>- Implement hybrid classification, where Bayesian filtering is refined by GPT.</li> <li>- Optimize GPT API response handling for lower latency and improved reliability.</li> <li>- Implement dynamic thresholds for switching between Bayesian and GPT classification.</li> </ul>
August 2 - August 8 <b>Week 10</b>	Full GPT Integration	<ul style="list-style-type: none"> <li>- Implement real-time adaptive learning, where GPT dynamically updates Bayesian classification.</li> <li>- Optimize API calls to reduce token usage and batch-process email classifications.</li> <li>- Compare API-based vs. self-hosted LLMs (LLaMA 3, Mistral-7B, Falcon-40B) for feasibility.</li> </ul>
August 9 - August 15 <b>Week 11</b>	Validation & Testing	<ul style="list-style-type: none"> <li>- Conduct real-world testing with a diverse email dataset.</li> <li>- Improve classification accuracy by fine-tuning BERT/RobERTa on email-specific data.</li> <li>- Benchmark Bayesian-only, GPT-only, and hybrid models based on accuracy, F1 score, and latency</li> </ul>
August 16 - August 22 <b>Week 12</b>	Documentation & Final Refinements	<ul style="list-style-type: none"> <li>- Document the final implementation of GPT integration in Rspamd.</li> </ul>

		<ul style="list-style-type: none"> <li>- Optimize performance benchmarks and record test findings.</li> <li>- Finalize code, API optimizations, and logging mechanisms.</li> </ul>
August 23 - August 30 <b>Week 13</b>	Final Evaluation Submission Prep	<ul style="list-style-type: none"> <li>- Prepare the final evaluation report with benchmarking results.</li> <li>- Ensure all deliverables are complete and reviewed before submission.</li> <li>- Complete any of the previous work that is pending.</li> </ul>
August 31 - September 7 <b>Week 14</b>	Final Submission & Mentor Review	<ul style="list-style-type: none"> <li>- Submit the final work.</li> <li>- Address mentor feedback and work on it if further modifications are required.</li> </ul>
September 2 - September 8 <b>Week 15</b>	Post-GSoC & Future Improvements	<ul style="list-style-type: none"> <li>- Explore additional improvements.</li> <li>- Wrap up documentation.</li> </ul>
September 9 - November 9	Extended Work (Optional)	<ul style="list-style-type: none"> <li>- Enhance autoencoder for better dataset generation and learning efficiency.</li> <li>- Explore additional self-hosted models for further cost optimization.</li> </ul>

## 5. Candidate Details

### 5.1 Why Am I Motivated by this project?

I have a strong interest in AI, ML, and software development, particularly in natural language processing and classification models. This project aligns with my previous experience in email classification with Smart Mail Guard. The requirements of the Multi-Class Bayesian Classifier match a lot with Smart Mail Guard. So I am pretty confident that I'll be able to not only complete the project but implement additional features and models too. The challenges I faced and the solutions I came up with to tackle those challenges will help me with rspamd too.

Link to **Smart Mail Guard**:

<https://github.com/aitwehrrg/SmartMailGuard>

### 5.2 Why Am I a Good Candidate?

I am a second-year (sophomore) undergraduate student at VJTI, Mumbai, pursuing a Bachelor of Technology degree in Computer Science. I have worked in the fields of machine learning and web development. I'm fluent in C, C++ and Python and Lua. I am an active member of the Community of Coders, a coding club in our university. We hold Machine Learning workshops and foster open source development.

- Experience with **Bayesian classification and multi-class email categorization** (Smart Mail Guard project).
- Proficiency in **Python, Machine Learning, and AI integration**.

### 5.3 Past Experience

- **Smart Mail Guard**

SmartMailGuard began as a binary classification project, initially using Naive Bayes to distinguish between spam and non-spam emails. This approach provided a basic solution, achieving an accuracy of around **97.55%** on the initial dataset. However, as the project evolved to require more complex classifications, I transitioned to Recurrent Neural Networks (**RNNs**) and Long Short-Term Memory (**LSTM**) models. RNNs, due to their forgetful behavior, struggled to capture contextual information, leading to a lower accuracy of **75.16%**. On the other hand, LSTM significantly improved performance by effectively capturing sequential dependencies within the email text, boosting accuracy to around **91.75%** and enabling the model to classify emails with more nuanced patterns.

As the project expanded to handle multiple categories, including "spam," "important," "finance," and "personal," I implemented Multinomial Naive Bayes, which worked well

for class-based frequency distributions, achieving an accuracy of approximately **82.03%**. To refine predictions and better handle imbalanced data, I then experimented with a combination of three machine learning algorithms: Support Vector Machine(SVM), Random Forest Classifier, and Decision Trees. These models provided a notable improvement, reaching an overall accuracy of **84%**. Finally, to address more complex text relationships, I incorporated **BERT** (Bidirectional Encoder Representations from Transformers), a state-of-the-art transformer model that further enhanced performance, pushing the classification accuracy to around **87.85%**.



- Contributor of [aeon-toolkit\(NumFOCUS\)](#)

<https://github.com/aeon-toolkit/aeon/pull/2516>

<https://github.com/aeon-toolkit/aeon/pull/2487>

<https://github.com/aeon-toolkit/aeon/pull/2612>

## 5.4 References

1.  Rspamd: fast opensource spam filter
2. Detailed [documentation of rspamd](#) to study the current implementation
3. Slides from the [presentations](#) of Rspamd by Vsevolod Stakhov
4.  Rspamd integration into FreeBSD.org mail infrastructure
5. **Tamboli, H.; Sarode, S.** An Effective Spam and Ham Word Classification Using Naïve Bayes Classifier. *Int. J. Sci. Res. Publ.* **2021**, *11*(7), 78–83.  
<https://doi.org/10.29322/IJSRP.11.07.2021.p11510>
6. Spam detection using LLMs:  
  
<https://ftp.fau.de/fosdem/2025/k4601/fosdem-2025-5114-enhancing-email-spam-detection-with-llms-practical-experience-with-rspamd-and-gpt.av1.webm>
7. [Rspamd's GitHub](#)

## 5.5 Availability

During the **Pre-GSoC phase**, I plan to dedicate approximately **4-5 hours per day** to research and familiarization with the Rspamd codebase, learning Lua and its scripting styles, and generating relevant datasets. This research phase will lay the foundation for a smooth transition into the GSoC coding period.

During the **GSoC Coding Period**, I am fully committed to the project with a **full-time availability of 6-8 hours per day**. This will allow me to focus on coding, testing, and refining the Multi-Class Bayesian Classifier, ensuring steady progress and timely milestones. I will be actively contributing to the project, collaborating with mentors, and making adjustments based on feedback.

After the **Post-GSoC phase**, I will remain available for **maintenance and future improvements**, continuing to support the project as needed, fix bugs, and implement any new features or enhancements based on feedback from the community or ongoing requirements.

## 6. Conclusion

This project aims to significantly enhance Rspamd's email classification capabilities by extending the existing Bayesian classification to handle multiple categories. By integrating adaptive AI-driven learning mechanisms, the system will become more robust, accurate, and scalable. My background in Machine Learning and Bayesian methods equips me with the necessary skills and expertise to implement this solution successfully. I am confident that I can contribute meaningfully to Rspamd's development and help improve its classification capabilities, making it more effective for a wider range of use cases.

# Thank you!