

# Algorithm Engineering CI/CD Telegram Bot

«Telegram Bot with automated compile checks,  
packaging, submission to OIOIOI platform and  
result check with comparison»

Tobias Veselsky



TU-Berlin, Faculty IV – Institut für Softwaretechnik und Theoretische  
Informatik, Algorithm Engineering  
Berlin, 06.02.2025

# Contents

<b>1</b>	<b>User Guide</b>	<b>1</b>
1.1	Overview of Features and Workflow . . . . .	1
1.2	Initial Setup and Configuration . . . . .	2
1.2.1	Starting the Bot . . . . .	2
1.2.2	Configuring the Bot with /setup . . . . .	4
1.2.3	Handling the submission_config.json File . . . . .	4
1.2.4	Providing the OIOIOI API Key via /config . . . . .	5
1.3	Repository Monitoring, Compilation, and Submission Workflow	6
1.3.1	New Commit Detection and Notification . . . . .	6
1.3.2	Compilation Check and Error Handling . . . . .	6
1.3.3	ZIP Packaging and Submission to OIOIOI . . . . .	8
1.3.4	Test Results Retrieval and Detailed Feedback . . . . .	9
1.3.5	Auto-Merge Functionality . . . . .	12
1.4	Group Chat Integration and Broadcast Features . . . . .	13
1.5	Sample submission_config.json File . . . . .	14
1.6	Command Reference . . . . .	16
1.7	Summary of a Typical User Session . . . . .	17
<b>2</b>	<b>Developer Guide</b>	<b>18</b>
2.1	Overview of Bot Architecture . . . . .	18
2.2	Core Modules and Their Responsibilities . . . . .	19
2.2.1	main.py . . . . .	19
2.2.2	config/config.py . . . . .	19
2.2.3	API Integration . . . . .	20
2.2.4	Git Operations (git_manager/git_operations.py) . . . . .	20
2.2.5	Compilation Handlers and the Compilation Manager . . . . .	20
2.2.6	Utility Modules . . . . .	22
2.3	Extending Language Support . . . . .	22
2.3.1	Steps to Add a New Language Handler . . . . .	22
2.4	Deployment and Environment Setup . . . . .	24
2.4.1	Dependencies . . . . .	24
2.4.2	Environment Variables . . . . .	24
2.4.3	Running the Bot . . . . .	24

# 1 User Guide

This chapter provides a comprehensive guide for end users of the Algorithm Engineering CI/CD Telegram Bot (AlgEngCIBot). It details the complete workflow—from starting the bot and performing the initial configuration to its continuous operation, submission of commits, and detailed reporting of test results along with improvement summaries. This guide assumes you have a basic understanding of Git and Telegram.

## 1.1 Overview of Features and Workflow

AlgEngCIBot automates the entire CI/CD process for your Algorithm Engineering projects. It continuously monitors your Git repository for new commits and automatically compiles and tests your code. Once your code passes the compilation checks, the bot packages your submission into ZIP files and submits it to the OIOIOI testing platform using its API. In addition, AlgEngCIBot retrieves detailed test results via web scraping (since the API does not provide an endpoint for results) and generates an Improvement Summary that compares the current test results with previous submissions, clearly highlighting improvements or regressions.

The complete workflow of AlgEngCIBot is divided into four main phases:

1. **Initial Setup and Configuration:** Configure the bot with your repository URL, primary branch, authentication method, and OIOIOI credentials.
2. **Repository Monitoring and Automated Submission:** The bot continuously monitors your repository for new commits on the branches specified in the `submission_config.json`. When a new commit is detected, it resets the repository to that commit, compiles the code using language-specific handlers, and – if the build is successful – packages the code into ZIP files and submits it to OIOIOI.

3. **Results Retrieval and Detailed Feedback:** After submission, the bot logs into the OIOIOI platform to fetch detailed test results via web scraping. It then sends you a comprehensive notification that includes the grouped test results, the overall score, and an Improvement Summary comparing the current results with previous submissions.
4. **Multi-Chat Integration for Team Collaboration:** In team settings, the bot can broadcast notifications to multiple Telegram chats. This allows the bot owner to add group chats to the broadcast list, ensuring that all team members receive the updates.

## 1.2 Initial Setup and Configuration

### 1.2.1 Starting the Bot

To start using AlgEngCIBot, open Telegram and search for its handle: @AlgEngCIBot. Alternatively, you can click the following link to open a conversation with the bot:

[@AlgEngCIBot](#)

Once you initiate the conversation by tapping "Start" or sending the /start command, the bot will respond with a welcome message that introduces its capabilities and provides an overview of the workflow.

**Example Interaction:**

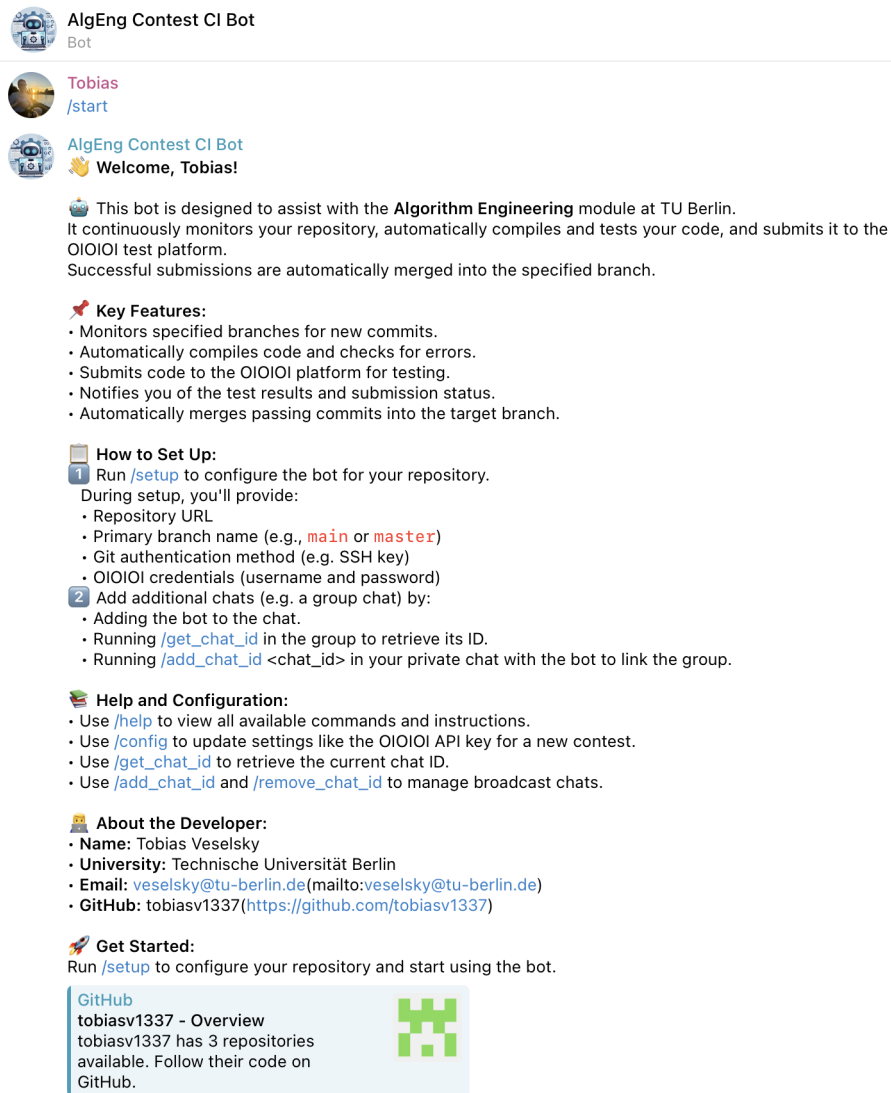
User: /start

Bot: "👋 Welcome, Alice!

This bot is designed to assist with the Algorithm Engineering module at TU Berlin.

It continuously monitors your repository, automatically compiles and tests your code, and submits it to the OIOIOI test platform. Successful submissions are automatically merged into the specified branch."

A sample screenshot is shown in [Figure 1.1](#).



The screenshot shows a Telegram chat window. At the top, the bot's profile is visible: a blue robot icon, the name "AlgEng Contest CI Bot", and the role "Bot". Below this, a user named "Tobias" has sent the command `/start`. The bot has responded with a welcome message. The message starts with a yellow robot icon and the text "Welcome, Tobias!". It then explains the bot's purpose: to assist with the Algorithm Engineering module at TU Berlin by monitoring repositories, compiling code, and submitting it to the OIOIOI test platform. It lists key features such as monitoring branches, automatic compilation, submission to OIOIOI, and automatic merging. It also provides setup instructions, including running `/setup` and adding additional chats. A "Help and Configuration" section lists commands like `/help`, `/config`, `/get_chat_id`, and `/add_chat_id`. An "About the Developer" section provides contact information for Tobias Veselsky. Finally, a "Get Started" section encourages running `/setup` and includes a GitHub repository overview for "tobiasv1337" with a green GitHub logo.

**AlgEng Contest CI Bot**  
Bot

**Tobias**  
`/start`

**AlgEng Contest CI Bot**  
🤖 **Welcome, Tobias!**

🤖 This bot is designed to assist with the **Algorithm Engineering** module at TU Berlin. It continuously monitors your repository, automatically compiles and tests your code, and submits it to the OIOIOI test platform. Successful submissions are automatically merged into the specified branch.

🔑 **Key Features:**

- Monitors specified branches for new commits.
- Automatically compiles code and checks for errors.
- Submits code to the OIOIOI platform for testing.
- Notifies you of the test results and submission status.
- Automatically merges passing commits into the target branch.

📖 **How to Set Up:**

- 1 Run `/setup` to configure the bot for your repository. During setup, you'll provide:
  - Repository URL
  - Primary branch name (e.g., `main` or `master`)
  - Git authentication method (e.g. SSH key)
  - OIOIOI credentials (username and password)
- 2 Add additional chats (e.g. a group chat) by:
  - Adding the bot to the chat.
  - Running `/get_chat_id` in the group to retrieve its ID.
  - Running `/add_chat_id <chat_id>` in your private chat with the bot to link the group.

📖 **Help and Configuration:**

- Use `/help` to view all available commands and instructions.
- Use `/config` to update settings like the OIOIOI API key for a new contest.
- Use `/get_chat_id` to retrieve the current chat ID.
- Use `/add_chat_id` and `/remove_chat_id` to manage broadcast chats.

👤 **About the Developer:**

- **Name:** Tobias Veselsky
- **University:** Technische Universität Berlin
- **Email:** [veselsky@tu-berlin.de](mailto:veselsky@tu-berlin.de) (<mailto:veselsky@tu-berlin.de>)
- **GitHub:** [tobiasv1337](https://github.com/tobiasv1337) (<https://github.com/tobiasv1337>)

🚀 **Get Started:**

Run `/setup` to configure your repository and start using the bot.

**GitHub**  
tobiasv1337 - Overview  
tobiasv1337 has 3 repositories available. Follow their code on GitHub.

**Fig. 1.1:** Welcome message after sending `/start`.

### 1.2.2 Configuring the Bot with /setup

To enable full functionality, you must configure the bot for your Git repository. Send the /setup command to start an interactive setup session. The process proceeds as follows:

1. **Terms and Conditions:**

The bot displays the terms and conditions. You must type accept to proceed (or abort to cancel).

**Sample Output:**



Terms and Conditions:

1. The bot owner is not responsible for any issues...

Type 'accept' to proceed or 'abort' to cancel.

2. **Repository URL:**

You are prompted to enter your repository URL.

**Important:** Use the HTTPS link without a trailing .git.

**Format:** https://github.com/yourusername/your-repo

3. **Primary Branch:**

Enter the name of the primary branch (usually main or master).

4. **Authentication Method:**

Choose one of the following methods:


- none: No authentication (public repo).
- https: You will be prompted for your Git username and password.
- ssh: You can either let the bot generate an SSH key pair or provide your own keys as plain text.

5. **OIOIOI Credentials:**

Provide your OIOIOI username and password. These credentials are used for logging in (via web scraping) to fetch test results from the OIOIOI website.

### 1.2.3 Handling the submission\_config.json File

After completing the setup, the bot clones your repository and verifies that your repository's primary branch contains a submission\_config.json file. If the file is missing, the bot will send an error message similar to:

” Error: Missing submission\_config.json in your primary branch.  
Please add the file to the repository and push the commit.”

**Action Required:** Add a valid submission\_config.json file to the root of your primary branch and push the commit. (See Section 1.5 for a sample file.)

### 1.2.4 Providing the OIOIOI API Key via /config

After adding the `submission_config.json` file, the bot will attempt to submit your latest commit. If no API key is found for the contest specified in your configuration, you will receive a message such as:

” Submission Aborted: No API key found for contest 'vc4'. Use /config to add the API key.”

To supply the missing API key:

- Send the `/config` command.
- When prompted, choose the option `OIOIOI_API_KEYS`.
- Follow the prompts to provide the contest ID and the corresponding API key.

## 1.3 Repository Monitoring, Compilation, and Submission Workflow

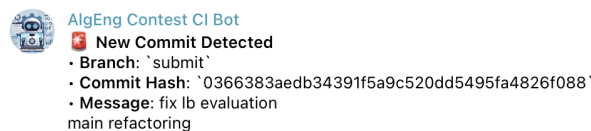
Once the bot is configured, it continuously monitors the specified branches for new commits.

### 1.3.1 New Commit Detection and Notification

When a new commit is detected:

- The bot fetches the latest commit hash and message.
- A notification is sent that includes:
  - The branch name.
  - The commit hash.
  - The commit message.

A corresponding screenshot is provided in [Figure 1.2](#).



**Fig. 1.2:** New commit notification received from the bot.

### 1.3.2 Compilation Check and Error Handling

After notifying you about a new commit, the bot proceeds to:

1. **Reset to Commit:**

The repository is reset to the specified commit.

2. **Compilation:**

The bot uses a language-specific handler (e.g., for Rust or C++) to compile your project.

- **Compilation Error Handling:** The bot follows rules set in `submission_config.json` for warnings and errors:
  - `ALLOW_WARNINGS`: Allows warnings if set to true.
  - `ALLOW_ERRORS`: Allows errors if set to true.
- If compilation is successful, you receive:  
Compilation Successful



- If compilation exceeds these limits, the bot sends a failure message as seen in [Figure 1.3](#).

#### ✗ Compiler Errors Detected in exact\_vc.zip

```

Updating crates.io index
Locking 16 packages to latest compatible versions
Adding rand v0.8.5 (available: v0.9.0)
Compiling proc-macro2 v1.0.93
Compiling unicode-ident v1.0.16
Compiling quote v1.0.38
Compiling libc v0.2.169
Compiling syn v2.0.98
Compiling zerocopy-derive v0.7.35
Checking cfg-if v1.0.0
Checking byteorder v1.5.0
Checking zerocopy v0.7.35
Checking getrandom v0.2.15
Checking rand_core v0.6.4
Checking ppv-lite86 v0.2.20
Checking rand_chacha v0.3.1
Checking rand v0.8.5
Checking lib v0.1.0 (/tmp/tmps1qxw4pu/lib)
error[E0583]: file not found for module `heuristics`
--> lib/src/lib.rs:6:1
|
6 | pub mod heuristics;
|   ^^^^^^^^^^^^^^^^^
|
= help: to create the module `heuristics`, create file "lib/src/heuristics.rs" or "lib/src/heuristics/mod.rs"
= note: if there is a `mod heuristics` elsewhere in the crate already, import it with `use crate::...` instead

For more information about this error, try `rustc --explain E0583`.
error: could not compile `lib` (lib) due to 1 previous error

```

#### ✗ Compilation Failed

```

• Branch: `submit`
• Commit Hash: `2b0f8ab758365dbe3f0579ecffcd21c730977606`
• Warnings Allowed: True
• Errors Allowed: False
Please review the compilation logs for more details.

```

**Fig. 1.3:** Notification for a compilation error.

### 1.3.3 ZIP Packaging and Submission to OIOIOI

Once the code compiles successfully:

1. **ZIP Packaging:**

The bot reads your `submission_config.json` file and creates one or more ZIP archives. These archives include the files and folders specified in the configuration.

2. **Submission:**

The ZIP files are submitted to the OIOIOI platform using its API.

- A confirmation message is sent upon a successful submission, for example:

Submission Accepted

- Branch: submit
- Submission ID: 98765

Waiting for results...

### 1.3.4 Test Results Retrieval and Detailed Feedback

After submission, the bot periodically checks the OIOIOI platform for test results. This involves:

- Logging into the OIOIOI website using your credentials.
- Web scraping the results page to extract detailed test results.

When results are available, you receive a comprehensive notification that includes:

- **Detailed Test Results:**

Grouped test outcomes for each test case including:

- Test name.
- Result status (e.g., OK, Failed, Skipped).
- Runtime.

- **Overall Score:**

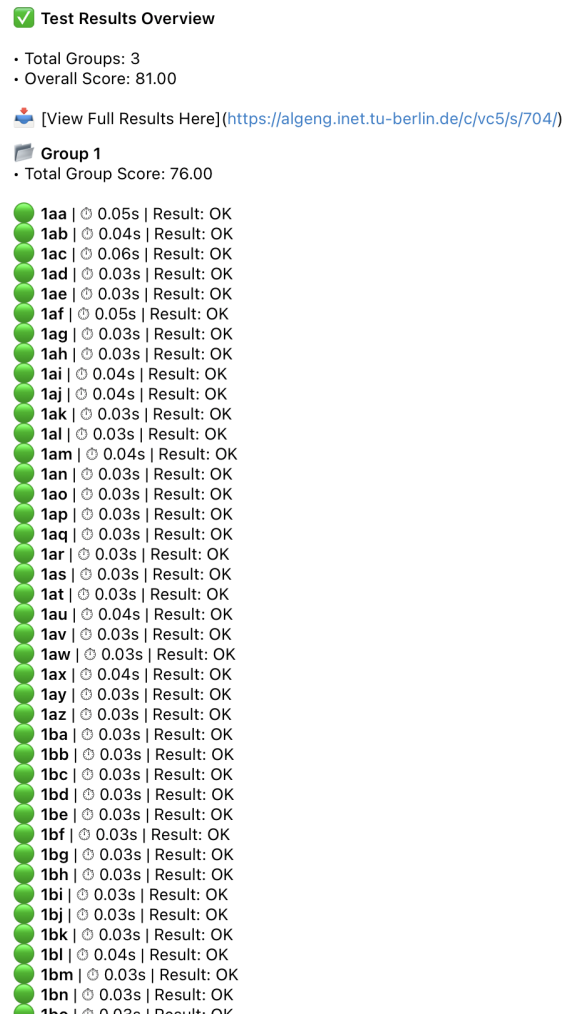
The combined score from all test groups.

- **Improvement Summary:**

A comparison with previous submissions, indicating:

- Additional tests passed or regressions.
- Changes in runtime performance.
- Group-specific details highlighting improvements or regressions.

A sample screenshot of a results notification is provided in Figures [1.4](#) (detailed view) and [1.5](#) (improvement summary).



**Fig. 1.4:** Detailed test results received via Telegram.

3cw | 0.00s | Result: SKIPPED  
 3cx | 0.00s | Result: SKIPPED  
 3cy | 0.00s | Result: SKIPPED  
 3cz | 0.00s | Result: SKIPPED  
 3da | 0.00s | Result: SKIPPED  
 3db | 0.00s | Result: SKIPPED  
 3dc | 0.00s | Result: SKIPPED  
 3dd | 0.00s | Result: SKIPPED  
 3de | 0.00s | Result: SKIPPED  
 3df | 0.00s | Result: SKIPPED  
 3dg | 0.00s | Result: SKIPPED  
 3dh | 0.00s | Result: SKIPPED  
 3di | 0.00s | Result: SKIPPED  
 3dj | 0.00s | Result: SKIPPED  
 3dk | 0.00s | Result: SKIPPED  
 3dl | 0.00s | Result: SKIPPED  
 3dm | 0.00s | Result: SKIPPED  
 3dn | 0.00s | Result: SKIPPED  
 3do | 0.00s | Result: SKIPPED  
 3dp | 0.00s | Result: SKIPPED  
 3dq | 0.00s | Result: SKIPPED  
 3dr | 0.00s | Result: SKIPPED  
 3ds | 0.00s | Result: SKIPPED  
 3dt | 0.00s | Result: SKIPPED  
 3du | 0.00s | Result: SKIPPED  
 3dv | 0.00s | Result: SKIPPED

**Final Summary**

- Total Groups: 3
- Overall Score: 81.00

[\[View Full Results Here\]\(https://algeng.inet.tu-berlin.de/c/vc5/s/704/\)](https://algeng.inet.tu-berlin.de/c/vc5/s/704/)

**Improvement Summary:**

- ✓ **Improvement:** 4 more tests passed! 🎉
- **Total Successful Tests:** 81
- **Runtime:** Slower by 1011.45s

**Group Details:**

- 🟡 Group 1: Same last solved test `1cx`.  
Runtime comparison: 🟡 No Change (0.13s → 0.13s)
- 🟢 Group 2: Improved. Last solved test: `2aa` → `2ae`
- 🟡 Group 3: No Changes. No tests solved in either submission.

[\[View Full Results Here\]\(https://algeng.inet.tu-berlin.de/c/vc5/s/704/\)](https://algeng.inet.tu-berlin.de/c/vc5/s/704/)

**⚠️ Auto-Merge Skipped**  
 Branch `submit` was not merged into `master` due to failed tests.  
 Tests Passed: 81/265

**Fig. 1.5:** Improvement summary received via Telegram.

### 1.3.5 Auto-Merge Functionality

AlgEngCIBot includes an optional auto-merge feature designed to assist your development workflow while keeping your primary branch safe. The bot monitors all branches listed under the `branches` key in your `submission_config.json` (e.g., `release`, `dev-a`, `dev-b`, etc.). However, it does not merge commits from all these branches automatically.

The auto-merge behavior is controlled by the `auto_merge_branch` key. This means that even though the bot monitors all branches listed in the `branches` key, only commits on the branch specified by `auto_merge_branch` are eligible for automatic merging into the primary branch.

For example, your `submission_config.json` might include:

**Lst. 1.1:** Partial `submission_config.json` snippet

```
{
  "branches": [ "dev-a", "dev-b", "submit" ],
  "auto_merge_branch": "release"
}
```

In this configuration, although the bot monitors commits on `dev-a`, `dev-b`, and `release`, only commits on the `release` branch are considered for auto-merging into your primary branch.

If you prefer to handle merges manually, you can disable the auto-merge feature by omitting the `auto_merge_branch` key or setting it to an empty value.

Note that the current implementation auto-merges as soon as all tests pass. This behavior may be refined in future updates to allow for more granular control over the merging process.

## 1.4 Group Chat Integration and Broadcast Features

AlgEngCIBot supports multi-user scenarios to facilitate team collaboration. A bot owner can broadcast notifications to additional Telegram chats (e.g., a group chat). The following commands are used:

`/get_chat_id` Retrieves the unique Telegram chat ID for the current chat.

The current chat ID is: -123456789

**Note:** You should add the bot to your additional chat and call this command in that chat to get its unique ID.

`/add_chat_id <chat_id>` Adds the specified chat (for example, a group chat) to the broadcast list so that notifications are sent there.

**Example:** `/add_chat_id -123456789`

**Note:** Use this command in your private chat with the bot to add a chat ID of another chat (e.g., your group chat).

`/remove_chat_id <chat_id>` Removes the specified chat from the broadcast list. **Note:** Use this command in your private chat with the bot to remove a chat ID.

`/list_chat_ids` Lists all chat IDs currently configured for receiving the bot's notifications.

## 1.5 Sample submission\_config.json File

A valid submission\_config.json file must reside in the root directory of your repository's primary branch and every commit to be handled in the branches that the bot should handle. If the file is missing, the bot will not process new commits.

Below is an example configuration along with explanations for each key:

**Lst. 1.2:** Sample submission\_config.json

```
{
  "language": "rust",
  "AUTOCOMMIT": true,
  "ALLOW_WARNINGS": true,
  "ALLOW_ERRORS": false,
  "contest_id": "vc4",
  "problem_short_name": "vc",
  "zip_files": [
    {
      "zip_name": "upper_bound.zip",
      "include_paths": [
        { "source": "upper_bound/src", "destination": "src" },
        { "source": "lib", "destination": "lib" },
        { "source": "upper_bound/Submission.Cargo.toml", "destination": "Cargo.toml" }
      ]
    }
  ],
  "branches": ["submit"],
  "auto_merge_branch": "submit"
}
```

- **language:** Specifies the programming language (e.g., rust or cpp). This determines the compilation handler.
- **AUTOCOMMIT:** If true, this commit will be handled by the bot.
- **ALLOW\_WARNINGS / ALLOW\_ERRORS:** Flags indicating whether warnings or errors during compilation are acceptable.
- **contest\_id & problem\_short\_name:** Identifiers for the OIOIOI contest and problem.
- **zip\_files:** An array defining ZIP packages, including which paths to include and their destination paths within the archive.



- **branches:** The branches that the bot monitors for new commits.
- **auto\_merge\_branch:** The branch into which passing commits are automatically merged.

## 1.6 Command Reference

`/start` Initiates a conversation with the bot. Displays the welcome message and a brief overview of the bot's capabilities.

`/setup` Begins the interactive setup process. Follow the prompts to:

- Accept the terms and conditions.
- Enter your repository URL.
- Specify the primary branch.
- Choose a Git authentication method.
- Provide OIOIOI credentials.

`/config` Update your configuration settings. Modify parameters such as:

- `repo_url`
- `primary_branch`
- `auth_method` (and related credentials)
- `OIOIOI_API_KEYS` (for contest-specific API keys)

**Note:** This command is especially useful if you need to supply a missing API key.

`/get_chat_id` Retrieves the unique Telegram chat ID for the current chat. Use this to obtain the ID of a group chat.

`/add_chat_id <chat_id>` Adds a chat (such as a group chat) to the broadcast list so that notifications are sent there.

`/remove_chat_id <chat_id>` Removes a specified chat from the broadcast list.

`/list_chat_ids` Lists all chat IDs currently configured for receiving the bot's notifications.

`/delete` Deletes all configuration data and repository files for the current chat. This action is irreversible.

`/abort` Aborts any ongoing setup or configuration process, resetting temporary states.

`/help` Displays a detailed help message with instructions for all available commands.

`/sample_config` Sends a sample `submission_config.json` template, including short explanations for each configuration field.

## 1.7 Summary of a Typical User Session

Below is an example scenario demonstrating the entire workflow:

1. **Initiation:** The user opens Telegram, searches for @AlgEngCIBot (or clicks the link [@AlgEngCIBot](#)), and sends the /start command to begin the conversation. The bot then sends a welcome message.
2. **Setup:** The user initiates /setup. After reviewing the terms and typing accept, the user provides:
  - Repository URL: `https://github.com/yourusername/your-repo`
  - Primary branch: `main`
  - Authentication method: `none`, `ssh` or `https`
  - OIOIOI username and password.

If the `submission_config.json` file is missing, the bot sends an error. The user adds the file and pushes a new commit.

3. **API Key Provision:** On the subsequent submission, if the API key for the contest (e.g. `vc4`) is missing, the bot prompts the user via an error message. The user then sends /config to supply the API key.
4. **Commit Monitoring and Submission:** With the configuration complete, the bot monitors the repository. When a new commit is detected:
  - A notification with commit details is sent.
  - The repository is reset to the new commit.
  - A compilation check is performed; on success, a confirmation is issued.
  - The code is packaged into ZIP files and submitted to OIOIOI.
5. **Results and Improvement Summary:** After submission, the bot logs into the OIOIOI website using your credentials and scrapes the test results. A detailed notification is sent that includes:
  - Grouped test results (status, runtime for each test).
  - Overall score.
  - An **Improvement Summary** that compares current results with previous submissions is sent.
6. **Group Collaboration:** The bot owner sends /get\_chat\_id in a group chat where the bot was added, to retrieve the group chat ID and then /add\_chat\_id <chat\_id> in the private chat to include the group in the broadcasting list. All team members now receive the same notifications.

## 2 Developer Guide

This chapter is intended for developers and maintainers of AlgEngCIBot. It explains the internal architecture, core modules, and the workflow implemented by the bot. In addition, it provides detailed instructions on how to extend the bot by adding support for additional programming languages.

### 2.1 Overview of Bot Architecture

AlgEngCIBot is built as an asynchronous Python application using `asyncio` and the `python-telegram-bot` library. Its architecture is designed around two concurrent tasks:

- **Telegram Task:** Listens for incoming commands and messages from users. It uses `telegram.ext.Application` to register command and message handlers.
- **CI Task Loop:** Periodically processes each user's repository by checking for new commits, running language-specific compilation checks, packaging the code into ZIP files, submitting the solution via the OIOIOI API, and finally fetching and parsing test results.

The bot's main entry point (in `main.py`) registers signal handlers for graceful shutdown, creates asynchronous tasks for both the Telegram task and the CI loop, and then runs them concurrently using `asyncio.gather`.

## 2.2 Core Modules and Their Responsibilities

### 2.2.1 main.py

This module is the entry point for AlgEngCIBot. It:

- Registers shutdown signals (e.g., SIGINT, SIGTERM) via `handle_shutdown_signal`.
- Creates and runs asynchronous tasks for both the Telegram bot and the CI task loop.

A key excerpt from `main.py` is shown below:

**Lst. 2.1:** Excerpt from `main.py`

```
async def main():
    bot_task = asyncio.create_task(telegram_task())
    ci_task = asyncio.create_task(ci_task_loop())
    await asyncio.gather(bot_task, ci_task)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, handle_shutdown_signal)
    signal.signal(signal.SIGTERM, handle_shutdown_signal)
    asyncio.run(main())
```

### 2.2.2 config/config.py

This module defines configuration parameters and loads environment variables. Examples include:

- `CHECK_INTERVAL`: The time (in seconds) between successive CI checks.
- `BACKOFF_TIME`: The delay imposed after encountering errors for a user.
- `OIOIOI_BASE_URL` and `TELEGRAM_BOT_TOKEN`: The base URL for OIOIOI and the Telegram Bot token, respectively.

### 2.2.3 API Integration

#### OIOIOI API (api/oioioi.py):

- Provides the OioioiAPI class to submit ZIP files to the OIOIOI platform.
- Uses the provided OIOIOI credentials to log in and then scrape the results page (because the API does not expose test result endpoints).
- Includes functions to fetch test results, generate result URLs, and wait for results.

#### Telegram API (api/telegram.py):

- Implements the TelegramBot class, which sends messages via the Telegram Bot API.
- Handles Markdown escaping and automatically splits long messages.

### 2.2.4 Git Operations (git\_manager/git\_operations.py)

This module handles all interactions with Git, including:

- Converting HTTPS URLs to SSH format.
- Generating SSH key pairs (using ssh-keygen).
- Tracking and saving the last commit hash (persisted in a JSON file).
- Executing Git commands in the repository (e.g., fetch, clone, checkout, and merge).

### 2.2.5 Compilation Handlers and the Compilation Manager

The bot's compilation workflow is modularized via a set of handlers located in the handlers directory.

#### Base Handler (handlers/base\_handler.py)

This module defines the abstract interface for language-specific compilation handlers. It includes:

- The abstract class LanguageHandler with an abstract static method compile(temp\_dir).
- The CompilationResult class, which encapsulates warnings and errors.
- The CompilationError exception class.

**Lst. 2.2:** Excerpt from base\_handler.py

```

from abc import ABC, abstractmethod

class CompilationResult:
    def __init__(self, warnings=None, errors=None):
        self.warnings = warnings or []
        self.errors = errors or []

class CompilationError(Exception):
    pass

class LanguageHandler(ABC):
    @staticmethod
    @abstractmethod
    def compile(temp_dir):
        """
        Perform a compilation check for the given project.
        Must return a CompilationResult or raise a CompilationError.
        """
        pass

```

**Language-Specific Handlers**

**Rust Handler** (handlers/rust\_handler.py): Implements the compile method by running cargo check in the temporary directory.

**C++ Handler** (handlers/cpp\_handler.py): Implements the compile method using CMake to configure and build the project. Both handlers return a CompilationResult or raise a CompilationError.

**Compilation Manager** (handlers/compilation\_manager.py)

This module coordinates the overall compilation process. It:

- Reads the language parameter from the user's configuration.
- Selects the corresponding language handler from the LANGUAGE\_HANDLERS dictionary.
- Iterates through the ZIP files generated for submission, extracting them to temporary directories and invoking the language handler's compile method.
- Sends detailed messages to the user via the Telegram API if errors or warnings are encountered.

### 2.2.6 Utility Modules

**File Operations** (utils/file\_operations.py): Handles configuration storage, repository paths, ZIP file creation according to the user's configuration, and deletion of obsolete data.

**Results Utilities** (utils/results\_utils.py): Formats test results, compares current submission results with historical data, and constructs improvement summaries.

**System Utilities** (utils/system.py): Contains just functions for handling shutdown signals.

**User Message Handler** (utils/user\_message\_handler.py): Registers Telegram command handlers and manages the interactive configuration process with the user.

## 2.3 Extending Language Support

One of the key design features of AlgEngCIBot is its modular approach to compilation. New language support can be added with minimal changes by implementing a new language handler.

### 2.3.1 Steps to Add a New Language Handler

1. **Create a New Module:** Create a new Python file in the handlers directory, for example, handlers/python\_handler.py, if you wish to add support for Python.
2. **Subclass the Base Handler:** Import LanguageHandler, CompilationResult, and CompilationError from handlers/base\_handler.py. Create a new class (e.g., PythonHandler) that subclasses LanguageHandler.
3. **Implement the compile Method:** Write the compile(temp\_dir) method to perform a syntax or compilation check for your target language. For example, you might use python -m py\_compile to check Python files. Return a CompilationResult instance or raise a CompilationError if errors occur.
4. **Register the Handler:** Update the LANGUAGE\_HANDLERS dictionary (in handlers/\_\_init\_\_.py) to include a new entry mapping the language string (e.g., "python") to your new handler class.



5. **Test the New Handler:** Commit a test change to a repository configured for Python, and verify that the bot uses your new handler correctly.

**Example:** Below is a minimal example for a hypothetical Python handler.

**Lst. 2.3:** Example PythonHandler implementation

```
from handlers.base_handler import LanguageHandler, CompilationResult,
    CompilationError
import subprocess
import os

class PythonHandler(LanguageHandler):
    @staticmethod
    def compile(temp_dir):
        # Attempt to compile all Python files to check for syntax errors
        for root, _, files in os.walk(temp_dir):
            for file in files:
                if file.endswith(".py"):
                    filepath = os.path.join(root, file)
                    result = subprocess.run(
                        ["python", "-m", "py_compile", filepath],
                        capture_output=True, text=True
                    )
                    if result.returncode != 0:
                        raise CompilationError(f"Syntax error in {file}: {
                            result.stderr}")
        return CompilationResult(warnings=["No warnings detected"])
```

After implementing the handler, register it in your central handler dictionary:

**Lst. 2.4:** Registering a new language handler

```
LANGUAGE_HANDLERS = {
    "rust": RustHandler,
    "cpp": CppHandler,
    "python": PythonHandler # New handler
}
```

## 2.4 Deployment and Environment Setup

### 2.4.1 Dependencies

All required dependencies are listed in requirements.txt. Key dependencies include:

- python-telegram-bot for Telegram integration.
- requests and beautifulsoup4 for web scraping and API interactions.
- asyncio for asynchronous task management.

### 2.4.2 Environment Variables

Environment variables (such as TELEGRAM\_BOT\_TOKEN) are loaded from a .env file via python-dotenv. Ensure that this ".env" file is present and properly configured in your deployment environment in the folder /config.

### 2.4.3 Running the Bot

Execute the bot by running main.py:

```
python main.py
```

The bot will initialize, set up signal handlers, and then concurrently run the Telegram and CI tasks.