Search for the $B \to K^* \nu \bar{\nu}$ decay at the Belle II experiment

Merna Abumusabh

1st year CSI report, 2024-2025

Supervisor(s): Jerome Baudot, Giulio Dujany

Keywords: Service Task, Calibration, Automated Pipelines

1. Introduction

The Belle II experiment operates at the SuperKEKB asymmetric-energy electron-positron collider in Tsukuba, Japan. As a B-factory, its primary scientific objective is to conduct precision studies in the flavor sector of the Standard Model (SM). This includes investigations into B and D mesons as well as tau leptons, with the aim of detecting signs of physics beyond the SM through rare decays, mixing effects, and other flavor-related phenomena.

The Belle II experiment offers a clean environment to study rare B-meson decays, providing a suitable ground for investigating flavor-changing neutral current (FCNC) processes, which in the SM proceed via loop processes and are an ideal place to look for BSM physics. These rare decays, such as $B \to K \nu \bar{\nu}$ and $B \to K^* \nu \bar{\nu}$, are theoretically clean due to the absence of non-factorizable hadronic effects, making them sensitive probes of new physics.

My thesis comprises two different topics. The first is focused on the **development and automation of the prompt calibration workflow** for Belle II, essential for ensuring the quality of the reconstructed data. In this context, we present the design and implementation of `b2luca`, a modern and scalable replacement for the existing prompt calibration system, `b2cal`. Each Belle II member performs a service task to become an author and this first part of my thesis correspond my service task. The second topic is the **search for the rare decay channel** $B \to K^* \nu \bar{\nu}$. In this document I will focus on the first topic.

2. The Calibration at Belle II

The calibration process in Belle II is structured into two main stages: the prompt calibration and the reprocessing. The prompt calibration generates the calibration constants - known as payloads - which are commonly stored in ROOT files. These payloads are later used during event reconstruction to correct for detector conditions. Since the detector's response can vary over time due to changing operational conditions, it is crucial to perform calibrations with sufficient time granularity to ensure accurate and reliable data reconstruction. The prompt calibration consists of several sequential steps that are summarized in the Fig 1 :

1. **Local calibrations**: some calibration constants are performed on data taken during special runs (for example determining pedestals) or describe the status of the detector (for examples channels that are turned off). The calibration responsibles provide those calibrations that by definition cannot be run in a central processing. In this step those calibrations are copied over to the central global tag used for prompt calibration.

2. **Raw calibrations**: Run on raw collision data all the calibrations that do not need tracks or that affect that results of the tracking (for example, the alignment and time calibration of the tracker detectors).

3. **cDST production**: Production of ROOT files that contain in addition to the raw event the result of the tracking. This step allows to speed up significantly the calibrations down the line as they won't need to run the tracking from raw data which takes a significant amount of time, often more than the calibration itself.

4. **cDST calibration**: Run on cDST files all the calibrations that require tracks but do not affect the result of the tracking (for example calibrations related to calorimeter and PID detectors).

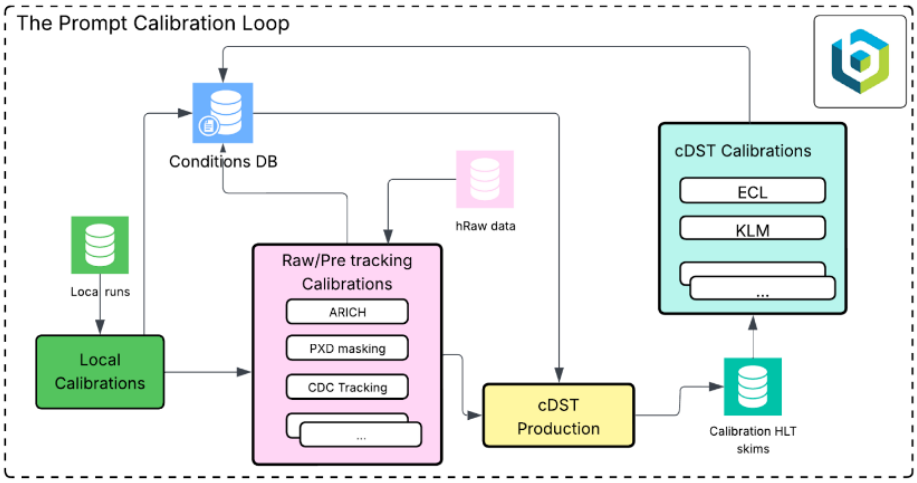

Figure 1: The prompt calibration loop

The oversight of prompt calibration is divided into two main components. First, the calibration tasks themselves are executed using the Calibration Framework (CAF), which handles job submission and output collection through dedicated CAF scripts. Second, the scheduling and coordination of these CAF scripts, including the management of task dependencies, ensures that the various calibration stages are executed in the correct order and with the necessary inputs.

The later aspect is the main focus of the service task, and it currently managed by the calibration framework `b2cal`, which automates the prompt calibration loop and offers Web-based user interface for monitoring and managing tasks. However, it is a complex system with many interconnected components introducing usability and maintenance challenges.

3. The new calibration framework

The new framework, `b2luca`, is built on b2luigi [1], an extension of the Luigi library originally developed by Spotify. Luigi is a Python-based tool that structures tasks into Directed Acyclic Graphs (DAGs), making task dependencies explicit and ensuring that operations are executed in the proper order. It also includes fault-tolerant features that prevent redundant task execution in case of interruptions.

Adapted specifically for the calibration requirements of the Belle II experiment, `b2luca` serves as a tool for handling all prompt calibration tasks at the Belle II Calibration Center, located at the Scientific Data and Computing Center (SDCC) at Brookhaven National Laboratory (BNL). The framework organizes and executes calibration workflows based on inter-task dependencies, enabling both parallel and sequential job execution as appropriate.

All calibration outputs are consolidated into a centralized database, promoting consistency, reproducibility, and easy retrieval. The system will include robust validation steps to guarantee the accuracy and reliability of the generated calibration constants. Unlike the earlier system that depended on a web interface, `b2luca` leverages GitLab for workflow management. GitLab's issue tracking and collaborative features streamline communication among calibration experts and enhance oversight of the entire process.

By unifying all calibration tasks under a single, DAG-based orchestration system, and by integrating GitLab for collaborative coordination, `b2luca` will hopefully offer a scalable, maintainable, and efficient solution to support the evolving calibration demands of the Belle II experiment.

4. b2luca workflow

`b2luca` itself is a command-line framework for automating prompt calibration at Belle II. It uses a modular, configuration-driven architecture governed by a single YAML file that defines all dynamic parameters—such as run ranges, input/output paths, and task settings. This structure ensures minimal hard-coding and enables seamless updates without modifying code. All configuration files are committed to GitLab.

The workflow of `b2luca` is as follows (All the points in each step are executed automatically with one command line):

1. **Configuration Setup:**
   The YAML configuration file defines core parameters, along with the calibration `JSON` setting files of each calibration, are committed to a GitLab repository named after the calibration bucket.
2. **Feedback Collection Phase:**
   Before calibration begins, the calibration responsibles review the settings of their calibrations and correct them if needed.
3. **Execution Phase:**
   - **Retrieve Updated Settings:**
     The latest calibration configurations are pulled from the GitLab repository, incorporating any recent changes.
   - **Generate Local RunDB:**
     A local CSV file is created by querying the Belle II online run database using run and experiment numbers defined in the YAML file.
   - **Calibration Loop:**
     The calibration loop follows the conventional order used in `b2cal`, with additional automation provided by `b2luca`:
     - **Local Calibrations:**
       Local calibration tasks primarily copy payloads to the global tag used for the prompt calibration. All are executed together via a `WrapperTask`[1].
     - **Raw Calibrations:**
       These tasks run hierarchically:
       * `CalibrationTask`: executes a single calibration script.
       * `GroupTask`: executes a group of calibration scripts that can run in parallel as they do not depend on each other.
       * `RunRawCalibrations`: Triggers running the whole series of raw calibrations.

       For each calibration scripts to run, there are two JSON files requirements that need to be presented, in which creating them is the first step in the `CalibrationTask` 3:
       (a) Generate expert configuration JSON file, parsing its content from the setup step.
       (b) Create an input data JSON file with run metadata.
       (c) Run the calibration script.
       (d) Create and execute validation scripts.
       (e) Log the task progress and final status (failure/success) to the dedicated GitLab issue, tagging the responsible user.
   - **cDST Production:**
     Produces derived cDST files from raw calibration outputs. In initial runs, only raw branches are selected. For repeated cDST production, input branches are updated to include refined constants.
   - **cDST Calibrations**: These follow the same execution flow as the raw calibrations.
   - **Logs Archiving:**
     Once a group of tasks is completed, the output files—excluding large intermediate data—are archived. Logs and results are stored in long-term storage systems, following a standardized naming convention based on the calibration bucket and execution date. A reference to the archive location is recorded in the corresponding GitLab summary issue for traceability.

5. Conclusion

The service task involving working on the new prompt calibration framework, `b2luca`, was officially validated. Work on testing, refining and extending its capabilities continues, including integrating additional validation checks and adapting the system for long-term maintainability.

[1] The main two classes in Luigi are the `Task`, and the `WrapperTask`. The latter runs multiple tasks in the same execution.

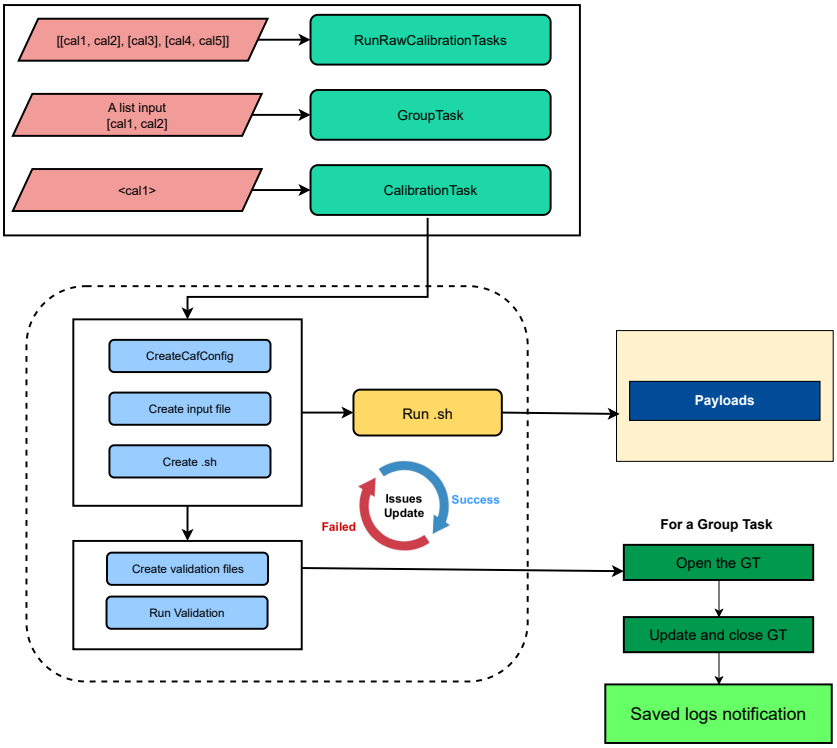

Figure 2: The implementation map of the `RunRawCalibrations` with the main steps inside the `CalibrationTask`

Some Belle II Jargon and abbreviations

- `b2cal`: The old Belle II prompt calibration framework.
- cDST: Calibrated Data Summary Table.
- GT: Global Tags — a collection of payloads and IoVs.
- IoV: Interval of Validity — Each payload is associated with a specific time range (in terms of experiment/run numbers) during which the conditions it describes were valid.
- Payload: ROOT files containing the calibration constants. These constitute the main calibration outputs.
- RunDB: Runs Database used to retrieve metadata for run numbers.

References

[1] b2luigi. `https://b2luigi.belle2.org/`.