76th International Astronautical Congress (IAC 2025), Sydney, Australia, 29 Sep-3 Oct 2025.

IAC-25,E2,3-GTS.4,8,x100719

An Integrated Test and Analysis Framework for Solid Propulsion: From Experimental Sounding Rockets to Emerging Space Applications

Junghyeon Yun[a]*, Yunseo Kim[b], Hyunwoo Park[a], Seockoo Chun[a], Surin Kim[a], Jooyong Yang[a], Jonghwan Yoon[a], Jiwan Seo[a]

[a] *Department of Aerospace Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, Republic of Korea 08826, slnstill3am@snu.ac.kr*
[b] *Department of Nuclear Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, Republic of Korea 08826*
* Corresponding Author

Abstract

This paper presents a reproducible, integrated framework for the test and analysis of solid rocket motors, developed to provide a reliable, low-cost propulsion test infrastructure for student teams, small companies, and research laboratories. The comprehensive system encompasses the design of a versatile static fire test stand, a custom data acquisition (DAQ) chain utilizing a load cell and pressure transducers, and a robust data processing methodology. The framework and its constituent systems have been rigorously validated through extensive use, incorporating lessons learned from over 30 static fire tests and supporting 11 sounding rocket flight missions. The principal result of this work is the development and validation of an end-to-end, science-based process that systematically addresses common measurement challenges, such as thermal drift, mechanical resonance, and noise artifacts, to accurately derive thrust, chamber pressure, and burn rate data. To accelerate progress and promote collaborative development within the broader aerospace community, the associated analysis software, the HANARO Static-Fire Toolkit (SFT), along with the hardware design and representative datasets, is being released as open-source software. This public dissemination aims to significantly lower the entry barrier for new practitioners in solid propulsion, enabling a shift from isolated competition toward cooperative progress in designing, testing, and operating propulsion systems for both experimental and emerging commercial space applications.

Keywords: Solid Propulsion, Integrated Framework, Thrust Measurement, Data Processing, Sounding Rocket, Open Source

Nomenclature

I-Class Motor	Total impulse 320–640 N·s
M-Class Motor	Total impulse 5120–10240 N·s

Acronyms/Abbreviations

IREC	International Rocket Engineering Competition
COTS	Commercial, Off-the-Shelf
SRAD	Student Researched and Developed
KNSB	Potassium Nitrate - Sorbitol
DAQ	Data Acquisition
HANARO SFT	HANARO Static-Fire Toolkit

1. Introduction

1.1 Background and Motivation

Hanaro, the student rocket team at Seoul National University with more than three decades of history, provides a platform where students cultivate fundamental engineering capabilities by designing and integrating rocket subsystems. In 2025, Hanaro conducted multiple flight missions, including its fourth participation in the IREC 10k COTS category, where the team achieved first successful launch and recovery of HARANG 2, while simultaneously executing an industrial payload sub-mission in collaboration with Korean Air.

A major factor in this achievement was a preliminary test flight of KHAOS in Korea. The KHAOS sounding rocket, which reached an apogee of 2 km, enabled full validation of the launch, recovery, and payload ejection sequence for HARANG 2 prior to the IREC mission. Since COTS motors are not available in Korea, we propulsion team designed and tested our own M-class motor, acquiring reliable data after several combustion tests to support flight operations of KHAOS rocket.

Because international shipment of solid propellant is difficult, propulsion systems for rockets from small experimental rockets and high-thrust sounding rockets were developed entirely by our team, with each motor undergoing static testing and data validation before launch. Through this process, we gradually established and refined a framework for propulsion system testing and data analysis.

In experimental sounding rocketry, solid motors remain the predominant propulsion option. Yet, university rocket teams are often compelled to design and operate their own propulsion systems, facing

substantial challenges due to the lack of integrated infrastructure for static testing and data analysis. These challenges were particularly evident among Korean universities participating in NURA competition, where securing reliable thrust data and ensuring reliable propulsion operations proved difficult.

To mitigate these challenges, we systematized and publicly released a thrust measurement system that bridges propulsion-system design and flight operations, and open-sourced the post-processing software used in our analyses. We also compiled and documented our operational case studies to assist student rocketry teams and other organizations that design their own solid propulsion units. By disseminating this system, we aim to lay the groundwork for low-cost propulsion test infrastructure not only for sounding rockets but also for small companies and laboratories for whom propulsion research is not a primary focus. This paper presents that effort.

1.2 Scope and Objectives

This paper documents reproducible, end-to-end framework for thrust measurement and data analysis method for design verification of sounding rocket solid propulsion system. The scope includes static test stand design, signal conditioning DAQ chain, and the post processing methods to derive accurate thrust, chamber pressure, and burn rate data. The system reflects more than 30 static fire tests conducted by Hanaro propulsion team and 11 flight missions executed by 2024 and the first half of 2025.

Main objectives of this paper are listed below

1. Describe the integrated system of thrust measurement and data analysis methods with sufficient detail and design intentions to enable direct reproduction.
2. Open source the analysis software "Hanaro Static Fire Toolkit" and provide guidelines so that diverse users can access, run, and extend the code.
3. Share operational case studies that illustrate the propulsion system validation process the propulsion team went through, thereby demonstrating practical deployment.
4. Lay the groundwork for low cost-propulsion test infrastructure applicable to future space-industry use cases

2. System Architecture Overview

Fig. 1 depicts the end-to-end workflow of the integrated test and analysis system, from static fire test and data acquisition to data post processing. The static test stand is designed for versatility, enabling mounting and ignition of solid motors across a range of sizes. During the static firing, thrust is measured with a load cell, chamber pressure with pressure transducer. To operate these sensors, the data-acquisition (DAQ) system supplies excitation, regulates input voltage, amplifies output signals, and provides real-time visualization of thrust.

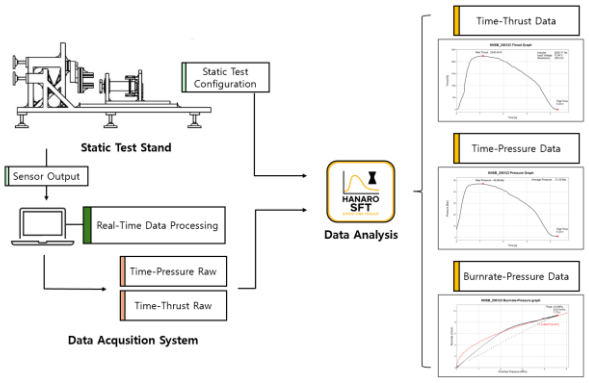


Fig. 1. Workflow of integrated static fire test and data analysis framework

Raw time-history data, together with configuration metadata, is subsequently processed with our Hanaro Static-Fire Toolkit. The resulting thrust, pressure, burn rate data acquired through this process is then used for design verification and operation. In the following sections, we describe the detailed design of integrated system's components.

3. Thrust Measurement System Design

3.1 Design Requirements

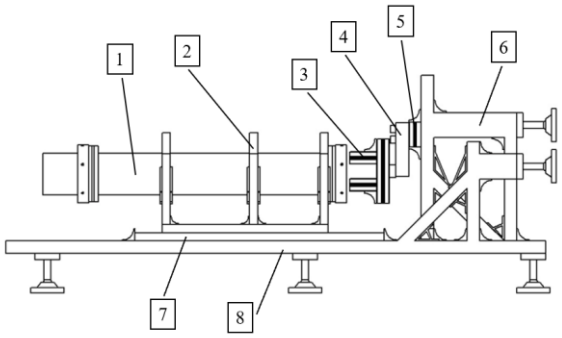
The thrust measurement system consists of a static test stand and a data acquisition system. These requirements below state what the product shall do in order to support static fire testing with reliable, repeatable measurements. As a student team without a permanent static test facility and without the resources to build a dedicated rig for each propulsion design, we defined the requirements below to enable effective static fire testing across a wide range of motor profiles.

1. The static test stand shall accommodate a range of motor diameters and lengths developed by the team.
2. The test stand shall provide mounting interfaces that restrain thrust, torque, and lateral loads from the motor.
3. The test stand shall support both forward-end ignition and after-end ignition.
4. The test stand shall enable nominal static fire operation in typical outdoor environment used by the team.
5. The DAQ shall record time-stamped thrust and chamber pressure data together with configuration metadata.

6. The measurement chain shall measure high thrust levels without signal saturation or clipping.
7. The measurement system shall be up at range and be ready to fire within 20 minutes and be operatable by a crew of eight or fewer.

3.2 Static Test Stand Hardware Design

The test stand is built primarily from aluminium profiles to increase stiffness while keeping cost and mass low, and to allow quick replacement of damaged parts. The assembly comprises the Lower Support Frame, Linear Guide (Rail) with Linear Bearing Blocks, Rear Support Frame, Load Cell Support Bracket, Load Cell with Load Cell Adapter, and Motor Mounting Frame. The V-type cradling of the motor is combined with low-friction sliding on the linear guide to improve measurement linearity and repeatability. The number of V cradles and upper clamps can be adjusted according to motor size.



No.	Components
1	Static Fire Test Motor
2	Motor Mounting Frame
3	Load Cell Adapter
4	Load Cell
5	Load Cell Support Bracket
6	Rear Support Frame
7	Linear Guide (Rail)
8	Lower Support Frame

Fig. 2. Static Test Stand Hardware Overview

3.2.1 Motor mounting frame

The Motor Mounting Frame supports the motor on V-shaped cradles, each cradle formed by two 20×20×165.46 mm aluminium profiles cut at 45°. The upper restraint is an inverted-V clamp that presses the motor down; this clamp is fabricated to match the specific motor outer diameter and is essential for rapid swaps across different motors. The frame rides on four Linear Bearing Blocks coupled to the Linear Guide (Rail), providing smooth axial motion and consistent alignment with the Load Cell Adapter. This configuration increases fixture repeatability and enables reliable thrust transfer to the load cell while keeping setup changes minimal across motor sizes.

3.2.2 Load cell adapter

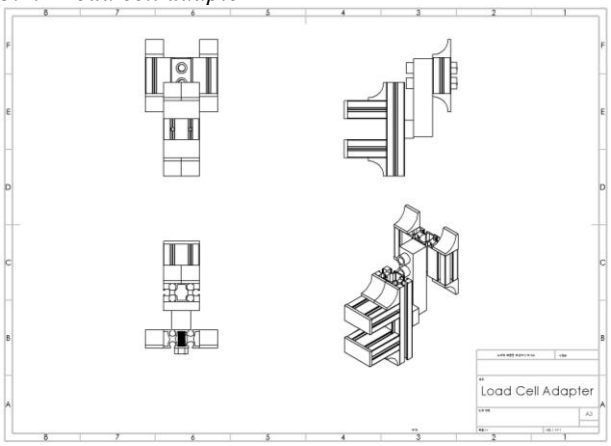


Fig. 3. Drawing of Load Cell Adapter

As shown in Fig. 3, the Load Cell Support Bracket and Load Cell Adapter were designed to allow free repositioning of the Load Cell so the stand can accommodate motors of different profiles. The bracket is mounted on the centreline of the Rear Support Frame and clamps to its two vertical columns. Fabricated from a 30×120×60 mm aluminium profile, it can be moved vertically when the clamping bolts are loosened and then locked in place. The Load Cell Adapter consists of one Foot Base 3060, one 30×60×155 mm aluminium profile, and two 30×65×60 mm aluminium profiles with 3060 aluminium profile cap. This configuration provides a stiff interface that minimizes load dispersion and delivers the axial force directly to the Load Cell.

3.2.3 Rear Support Frame

The Rear Support Frame is designed to sit flush against a wall so that the reaction force is transferred directly into the wall. Four vertical columns of 30×30×240 mm aluminium profiles are arranged symmetrically at the ends, and a 30×60×90 mm profile is attached to the back of each end column. A 30×30×400 mm vertical member for the load-cell interface is placed between the two centre columns and tied with a 30×60×200 mm profile with two 60x60 flat brackets on each column. For global adjustment, four Foot Base 3060 units and ball joint levelling feet are installed, allowing precise alignment with respect to the wall. To increase thrust-load capacity, 30×30×240 mm profiles cut at 45° are added to the front faces of the two end columns and at the centre connection. This reinforcement alleviates the bolt-capacity shortfall

identified in the previous stand, and it simplifies the load transfer path from wall - frame - load cell, reducing thrust-signal distortion and sensitivity to installation variability.

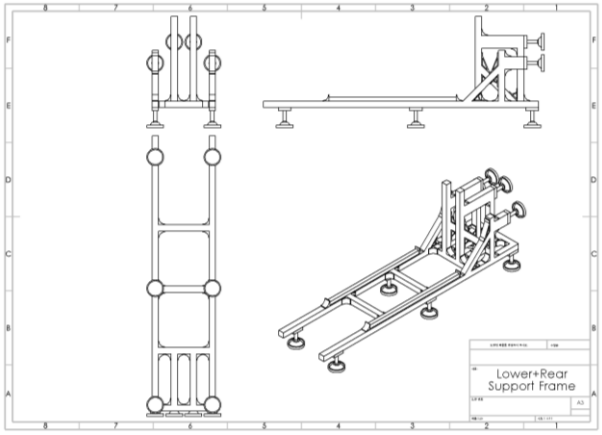


Fig. 4. Drawing of Rear and Lower Support Frame

3.2.4 Lower Support Frame

As shown in Fig. 4, the Lower Support Frame uses two 30×30×1300 mm aluminium profiles as the main rails, spaced 240 mm apart and cross-braced by two 30×30×240 mm profiles placed at equal intervals. Three additional 30×30×240 mm members form the interface to the Rear Support Frame. To accommodate uneven test-site floors, six Foot Base 3060 units with six ball joint levelling feet are installed on the underside, enabling rapid levelling before each test. The Linear Guide (Rail) is fastened directly on top of the lower frame, and the Motor Mounting Frame rides on it. This arrangement reduces friction in the thrust direction and blocks alternate load paths, improving measurement repeatability.

3.3 Data Acquisition System and Data Flow

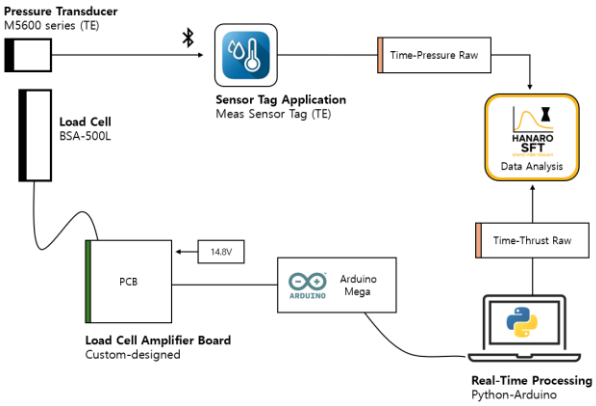


Fig. 5. Data Acquisition System Overview

The data acquisition system uses two sensors: a load cell for thrust and a pressure transducer for chamber pressure. The load cell is mounted on the Load Cell Support Bracket of the test stand, and the pressure transducer is plumbed to the motor bulkhead via a stainless-steel tube and fitting.

We typically used BSA-250L and BSA-500L load cells for our testing; the capacity can be changed as needed. When the capacity is changed, the "rated_load" value in the post-processing configuration must be updated accordingly.

The load cell connects to a custom Load Cell Amplifier Board (PCB) that accepts a 14.8 V supply, regulates a stable 12V excitation for the bridge, amplifies the output signal, and provides manual zero adjustment. The amplified analog signal is digitized through the Arduino Mega and streamed over serial to a Python script for real-time visualization. Raw thrust data are logged to CSV at approximately 1 kHz. We originally used an NI DAQ module, but recurring licensing issues, reliability problems with aging hardware, and the desire for portability and open-source release motivated the current Python–Arduino architecture.

For pressure, we employ a Bluetooth model paired with a smartphone sensor-tag application; raw pressure data are recorded at 10 Hz and exported for post-processing. The implications of the different sampling rates and the synchronization methods are discussed in the next section.

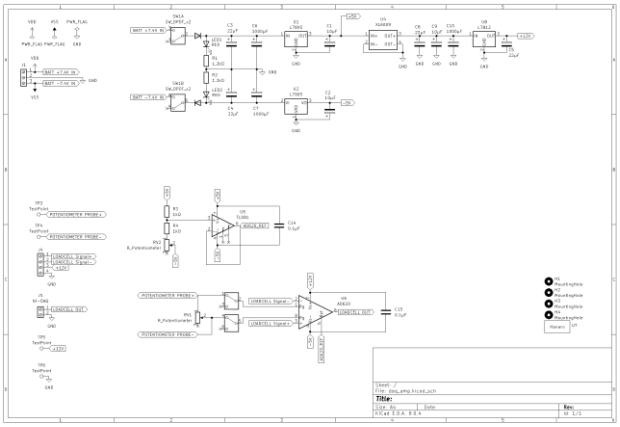


Fig. 6. Circit Diagram of Load Cell Amplifier Board

The Load Cell Amplifier Board (PCB) serves two functions: it regulates the reference/excitation voltage supplied to the load cell, and it outputs an amplified version of the load-cell signal. As shown in the upper portion of Fig. 6, the board receives an external 14.8 V input through basic protection circuitry and regulates it to 12 V for the load cell. In the lower portion of the schematic, a resistive divider and a voltage follower

generate a reference voltage (−5 V to +5 V range), and an instrumentation amplifier (AD620) uses this reference to amplify the load-cell output. The gain is set with a potentiometer, and the reference node is used to trim the zero so that the output is 0 V at no load; the amplified signal is then provided at the board output. Based on this circuit, we designed and fabricated the PCB shown in Fig. 7. For operational convenience, the layout includes test points to measure the gain-setting resistor and the load cell excitation voltage.

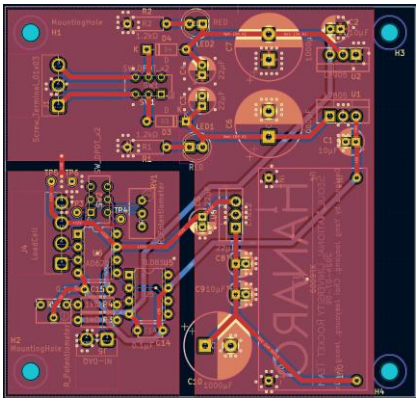


Fig. 7. PCB Design of Load Cell Amplifier Board

4. Data Processing Methodology

4.1 Thrust Data Processing and Filtering

We model the load-cell/DAQ measurement chain as a band-limited system subject to thermal drift, mechanical resonance, and quantization artifacts. The thrust measurement system employs a 500 kgf rated capacity load cell with 3 mV/V sensitivity, conditioned through a differential amplifier with 49.4 kΩ of internal resistance, configurable gain resistance (typically around 200 Ω) and 12 V excitation voltage. Raw voltage signals are converted to thrust values using calibrated parameters that account for bridge sensitivity, excitation voltage, and amplifier characteristics.

The measurement chain introduces several sources of distortion that require systematic correction. Thermal drift manifests as gradual baseline shifts due to temperature-dependent strain gauge resistance and elastic modulus variations. Additionally, DAQ quantization and sampling artifacts create impulsive noise requiring robust filtering strategies. Our filtering pipeline addresses these artifacts through a multi-stage approach prioritizing signal integrity over computational efficiency.

$$\text{gain}=\text{offset}_{\text{gain}}+R_{\text{internal}}\ [\Omega]\}/R_{\text{gain}}\ [\Omega]$$
$$V_{\text{bridge}}\ [\text{mV}]\ =\ \text{sensitivity}\ [\text{mV/V}]\ \times V_{\text{excitation}}\ [\text{V}]\ \times\text{thrust}\ [\text{N}]\ /\ (\text{capacity}\ [\text{kgf}]\ \times\ g\ [\text{N/kgf}])$$
$$V_{\text{measured}}[\text{V}]=\left(V_{\text{bridge}}/1000\right)[\text{V}]\times\text{gain}$$
$$\text{thrust}[\text{N}]=\left(V_{\text{bridge}}/\text{sensitivity}\right)[\text{V}]/V_{\text{excitation}}\times(\text{capacity}\times g)[\text{N}]$$

Eqn. 1. Voltage-to-Thrust Conversion Formula

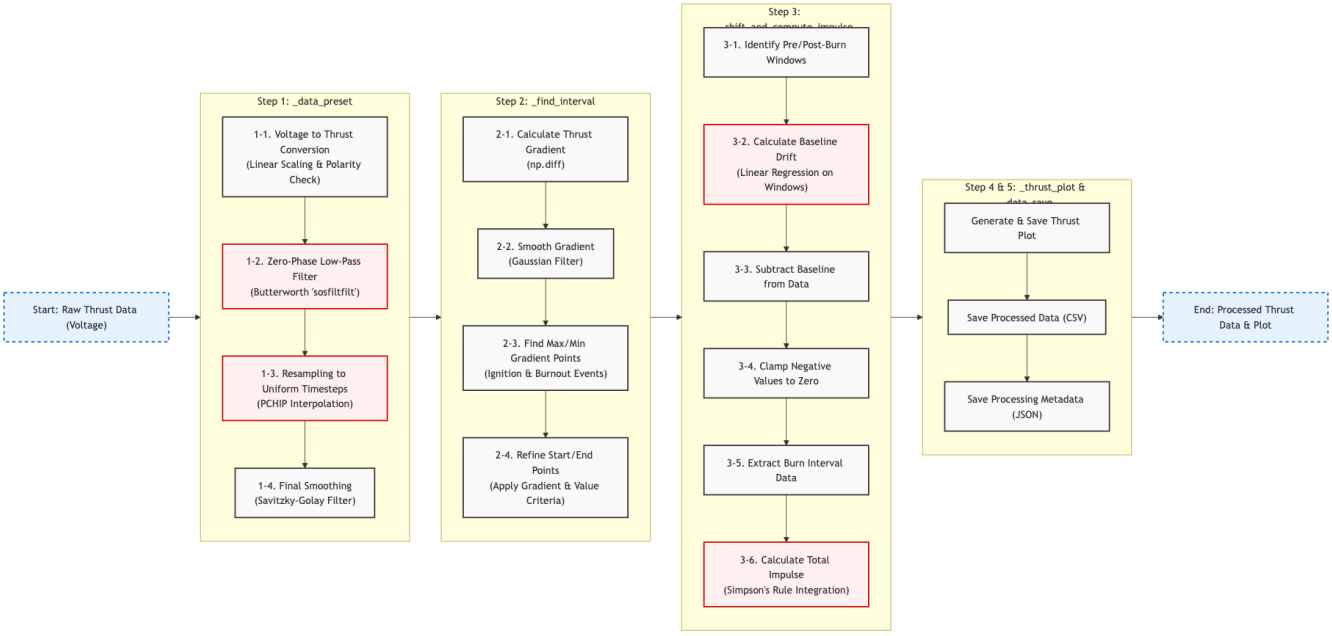


Fig. 8. Flowchart of the Thrust Data Processing Pipeline

4.1.1 Data Pre-processing

The thrust data preprocessing pipeline (_data_preset) implements a four-stage approach designed to address systematic measurement artifacts while preserving genuine combustion dynamics. The preprocessing stage applies sequential transformations to raw voltage measurements.

(1) Voltage-to-Thrust Conversion

Raw DAQ voltages undergo linear scaling based on calibrated load cell parameters, incorporating bridge sensitivity, amplifier gain, and excitation voltage. The conversion includes polarity verification to ensure correct thrust direction, with automatic correction for reversed sensor connections commonly encountered during test setup.

(2) Zero-Phase Low-Pass Filtering

We apply a 5th-order Butterworth low-pass filter using second-order section (SOS) implementation via 'scipy.signal.sosfiltfilt' [1,2,3]. This zero-phase filtering approach eliminates group delay while providing -100 dB/decade roll-off beyond the cutoff frequency. The sosfiltfilt implementation performs forward-backward filtering, effectively doubling the filter order to 10th-order while maintaining zero phase shift—critical for preserving ignition timing accuracy.

(3) Uniform Resampling

Raw data with irregular time steps undergoes resampling to uniform 10 ms intervals (100 Hz) using PCHIP (Piecewise Cubic Hermite Interpolating Polynomial) interpolation. PCHIP preserves monotonicity and avoids overshooting common in cubic spline interpolation, particularly important near ignition and burnout transients [1,4,5]. To prevent aliasing during the down sampling/resampling process, PCHIP interpolation is performed after passing through the LPF.

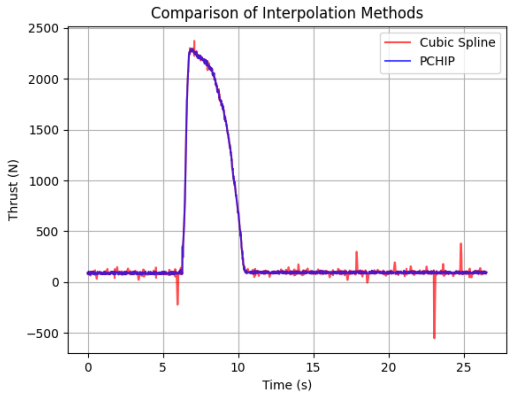


Fig. 9. Comparison of interpolation results using Cubic Spline (red) and PCHIP (blue)

Fig. 9 shows Cubic Spline is sensitive to outliers, leading to overshooting issues when sudden changes occur due to vibrations or noise. PCHIP, on the other hand, ensures monotonicity, making it suitable for thrust data processing.

(4) Savitzky-Golay Smoothing

Final smoothing employs a Savitzky-Golay filter [1,6,7] with 11-point window length and 3rd-order polynomial fitting. This approach provides superior noise reduction for subsequent derivative calculations while preserving peak characteristics essential for meaningful interval detection and impulse calculation.

4.1.2 Meaningful Interval Detection

Combustion window identification (_find_interval) employs gradient-based analysis rather than simple threshold methods. This approach provides robust event detection even under varying motor performance characteristics.

(1) Gradient Calculation and Smoothing

Thrust gradients are computed using 'numpy.diff()' [8,9], followed by strong Gaussian smoothing [1,10] (with σ=10) to reduce noise-induced artifacts in derivative calculations. The smoothed gradient profile enables reliable identification of ignition and burnout events through extrema detection.

(2) Event Detection Criteria

The algorithm identifies ignition and burnout points by locating maximum and minimum gradient values, respectively. Subsequently, refined start and end points are determined using dual criteria: 1) gradient magnitude falling below a specified fraction of the maximum gradient, and 2) simultaneous thrust magnitude dropping below 5% of maximum thrust. To enhance robustness against transient fluctuations such as ripple, which can create multiple local peaks, the algorithm implements a 0.8-second look-ahead/behind search window. This prevents premature termination of the search by verifying that a candidate point is not immediately followed (or preceded) by another significant gradient event, making the overall interval estimation resilient to multi-peaked thrust profiles. This combined approach ensures robust interval boundary detection across varying motor configurations and performance levels. Particularly, this combined methodology has been verified to enable robust interval estimation for unstable combustion phenomena like *Chuffing*, which are difficult to process automatically using gradient-based methods alone.

Fig. 10 shows the robustness of the interval estimation and processing even for data with multiple peaks and local maxima.

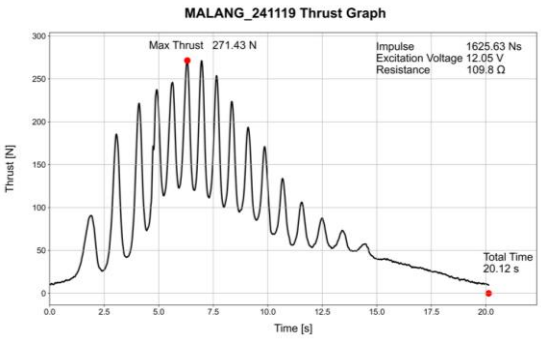


Fig. 10. Example of processing thrust data with chuffing patterns

4.1.3 Baseline Correction and Impulse Integration

(1) Baseline Drift Estimation

Pre-burn and post-burn windows are identified outside the meaningful interval for baseline characterization. Linear regression analysis on these windows quantifies systematic drift rates, typically manifesting as gradual offset changes due to thermal effects in the load cell and signal conditioning electronics.

(2) Drift Correction and Clamping

The estimated linear drift is subtracted from the entire time series, followed by clamping of negative thrust values to zero. This clamping operation prevents unphysical negative thrust readings while preserving the integrated impulse accuracy.

(3) Total Impulse Calculation

Total impulse computation employs Simpson's rule integration [1,11] over the extracted burn interval, providing superior accuracy compared to trapezoidal methods for the smooth thrust profiles typical of solid rocket motors.

4.2 Pressure Data Processing and Filtering

We utilize a pressure sensor with a low sampling rate of 10Hz. Chamber pressure data is interpolated synchronously with thrust data at 100 Hz. The pressure measurement chain exhibits different noise characteristics compared to thrust, with primary artifacts arising from thermal drift rather than mechanical vibration. Our pressure processing pipeline focuses on temporal synchronization with thrust measurements rather than frequency-domain filtering, reflecting the inherently smoother nature of pressure evolution compared to mechanical thrust measurements.

4.2.1 Data Pre-processing

The pressure preprocessing stage (_data_preset) handles format standardization and basic conditioning while avoiding over-filtering of genuine pressure dynamics.

(1) Datetime Parsing

Raw pressure data accommodates various timestamp formats through flexible parsing routines that handle both absolute timestamps and relative time vectors. The system automatically detects ISO 8601 format conventions and converts to uniform relative time representation.

(2) Optional Low-Pass Filtering

While our static-fire test data processing library supports the application of a low-pass filter to pressure signals, this stage was intentionally bypassed in the present study. The pressure sensor employed has a low native sampling rate (10 Hz), limiting the signal bandwidth and making additional filtering unnecessary. For teams utilizing higher-rate pressure transducers where noise is a concern, the same 5th-order zero-phase Butterworth filter (sosfiltfilt)[1,2,3] can be applied, though a relaxed cutoff frequency is recommended to preserve genuine pressure oscillations that may contain combustion stability information.

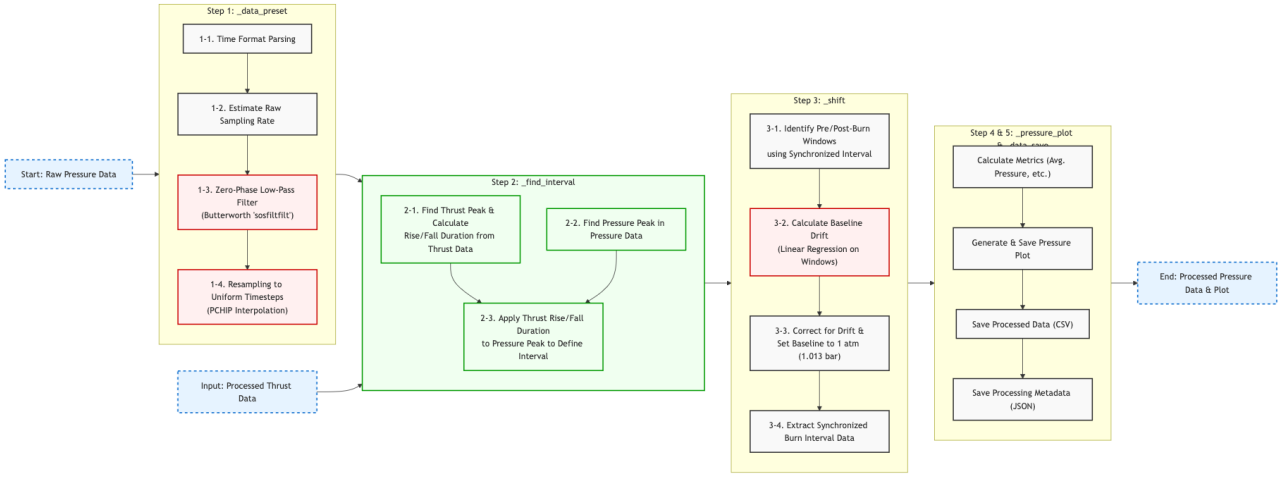


Fig. 11. Flowchart of the Pressure Data Processing Pipeline

(3) Uniform Resampling

Uniform up sampling to 100 Hz using PCHIP maintains temporal correspondence with thrust measurements while preserving pressure profile monotonicity through shape-preserving interpolation [1,4,5].

4.2.2. Interval Synchronization with Thrust Data

The critical point in pressure processing lies in synchronization with independently determined thrust intervals (_find_interval), ensuring consistent combustion window definitions across measurement modalities.

(1) Thrust-Derived Timing Parameter

The algorithm extracts the index corresponding to the peak thrust from the processed thrust data. This index serves as the primary temporal anchor, which establish the temporal framework for the subsequent pressure analysis.

(2) Pressure Peak Identification

Independent pressure peak detection identifies maximum chamber pressure occurrence, which serves as the temporal anchor point for interval synchronization.

(3) Interval Synchronization

Thrust-derived rise and fall durations are applied relative to the pressure peak to define consistent combustion windows. This approach ensures that pressure and thrust analyses operate on temporally aligned data segments, critical for subsequent burn rate analysis requiring both measurements.

4.2.3. Pressure Baseline Correction

Pressure baseline correction addresses atmospheric pressure variations and sensor drift while maintaining absolute pressure accuracy required for burn rate calculations.

(1) Baseline Drift Estimation

Pre- and post-burn baseline windows are established using the thrust-synchronized interval boundaries, ensuring consistent reference periods across all measurement channels.

(2) Drift Correction and Clamping

The estimated linear drift is subtracted from the entire time series, followed by clamping of negative thrust values to zero. This clamping operation prevents unphysical negative thrust readings. In addition, the gauge pressure normalized to standard atmospheric pressure (1.013 bar) is stored alongside the absolute pressure. This normalization enables consistent performance comparisons across different test sessions and environmental conditions.

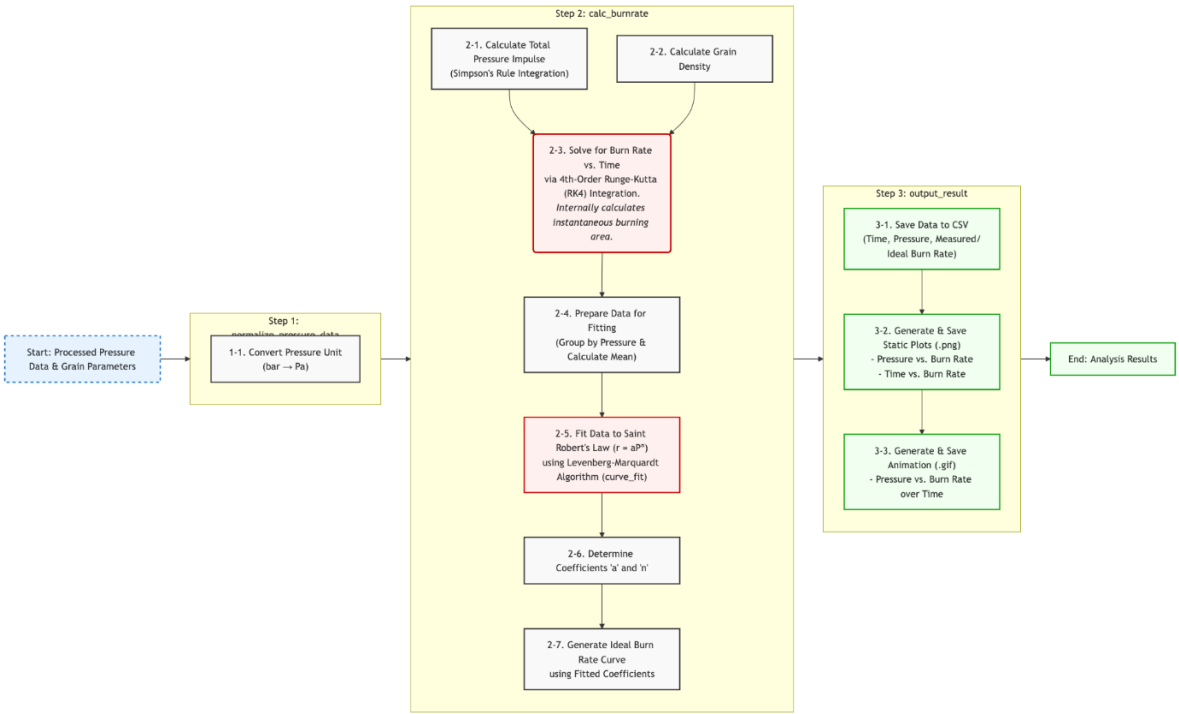


Fig. 12. Flowchart of the Burn Rate Estimation Pipeline

4.3 Burn Rate Estimation

This section details the computational methodology used to characterize the propellant's combustion properties from experimental pressure data. The primary objective is to determine the parameters of Saint-Robert's law (a.k.a. Vieille′s law), $r = aP^n$, where r is the linear burn rate (mm/s), P is the chamber pressure (Pa), and a and n are the burn rate coefficient and pressure exponent, respectively [12]. Our approach combines processed pressure measurements with BATES grain geometry parameters, employing numerical integration and nonlinear regression to extract these key propellant characteristics. The implementation leverages the fundamental relationship between chamber pressure evolution, instantaneous burning surface area, and propellant consumption rate during motor operation.

4.3.1 Pressure Data Normalization

The burn rate analysis pipeline begins with pressure data normalization to ensure consistent units and reference conditions for subsequent thermodynamic calculations.

(1) Unit Conversion

Processed absolute pressure data in bar units undergoes conversion to Pascal (Pa) for compatibility with SI-based thermodynamic relationships. This standardization ensures dimensional consistency throughout the burn rate calculation pipeline.

4.3.2 Numerical Burn Rate Calculation

The core burn rate calculation (calc_burnrate) implements a sophisticated numerical approach that solves the coupled problem of pressure evolution and burning surface area progression through time-domain integration.

(1) Pressure Impulse Integration

Total pressure impulse calculation employs Simpson's rule integration [1,11] over the synchronized burn interval, providing a fundamental performance metric while establishing the integration framework for subsequent calculations. The Simpson's method offers superior accuracy compared to trapezoidal approaches.

(2) Grain Density Calculation

Propellant density is computed from grain geometry parameters and total propellant mass, assuming the propellant grain to be a homogeneous mixture. This density serves as a critical parameter linking regression distance to volumetric burn rate calculations.

(3) Burn Rate Computation via RK4

The burn rate profile is determined by numerically solving the governing nonlinear differential equation for the regressed distance. Specifically, the 4th-order Runge-Kutta (RK4) method is employed to compute an approximate solution, yielding the burn rate at each discrete time step in the series. The RK4 approach is selected for its excellent stability and accuracy in handling the nonlinear dynamics inherent in solid rocket motor operation.

4.3.3 Saint-Robert's Law Parameter Extraction

The parameter extraction process employs robust nonlinear regression techniques to fit experimental burn rate data to the Saint-Robert's law relationship while accounting for measurement uncertainties and computational artifacts.

(1) Data Preparation and Grouping

Raw burn rate calculations undergo preprocessing to improve regression stability. Data points are grouped by pressure magnitude and averaged within each group to reduce scatter while preserving the fundamental pressure-burn rate relationship.

(2) Levenberg-Marquart Fitting

Saint-Robert parameter estimation utilizes the Levenberg-Marquardt algorithm via `scipy.optimize. curve_fit` for robust nonlinear least-squares regression [1,13]. The Levenberg-Marquardt method provides superior convergence properties compared to simple OLS (Ordinary Least Squares) linear regression alone.

The fitting process determines coefficients a and n in the Saint-Robert relationship $r = aP^n$. To enhance convergence speed and stability, initial parameter estimates are derived from an Ordinary Least Squares (OLS) regression on the logarithmically transformed version of the law ($\log r = \log a + n \log P$). This data-driven approach provides a robust starting point for the nonlinear fitting algorithm, which ensures reliable parameter extraction.

(3) Ideal Curve Generation

Using the fitted coefficients, the algorithm generates an ideal burn rate curve spanning the pressure range encountered during motor operation. This theoretical curve enables direct comparison with experimental data and provides the basis for performance prediction in scaled motor designs.

4.3.4 Results Validation and Output

The analysis pipeline produces comprehensive output data and visualizations designed to facilitate both immediate assessment and detailed subsequent analysis.

(1) Comprehensive Data Export

Results are saved in CSV format containing synchronized time series of pressure, measured burn rate from RK4 integration, and ideal burn rate from Saint-Robert fitting. This complete dataset enables external validation and alternative analysis approaches.

(2) Static Visualization

The system generates pressure vs. burn rate plots and time vs. burn rate plots in PNG format, providing immediate visual assessment of fitting quality and burn rate progression characteristics. These static plots facilitate rapid evaluation of motor performance and identification of anomalous behaviour patterns.

(3) Dynamic Animation

Animated GIF generation shows pressure vs. burn rate relationship evolution over time, providing intuitive visualization of the motor's operating trajectory through the Saint-Robert parameter space. This dynamic representation proves particularly valuable for identifying transient effects and verifying the consistency of the fitted relationship throughout the burn.

5. Motor Design and Static Fire Cases

To demonstrate applicability and discuss extensibility of the integrated framework, we present three applications: (5.1, 5.2) propulsion system validation for sounding rockets launched in 2024 and the first half of 2025, and (5.3) an ongoing AN-based solid-propellant development effort by the Hanaro propulsion team. For each case, we outline the project, summarize the motor design and test configuration, and present the outputs produced by the integrated framework.

5.1 Identity 3: A Versatile Sounding Rocket for Multi-Mission Capability

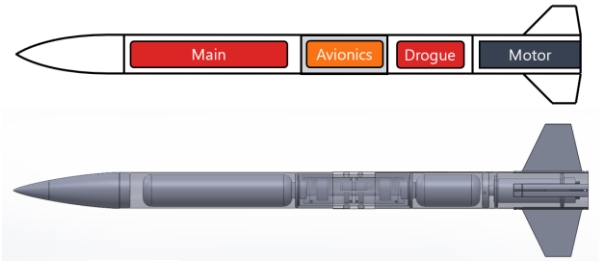


Fig. 13. Identity 3 Subsystem Overview

Identity 3 is Hanaro′s standardized, general-purpose sounding rocket, completed in June 2024 and continually operated to date. Designed as a compact platform to validate dual-parachute deployment and to support training of Hanaro′s freshmen, it documents the full design-to-manufacture process as a reference model for the team. More recently, the Identity 3 baseline has been used as a low altitude testbed to verify subsystems, such as canards and ejectable payloads.

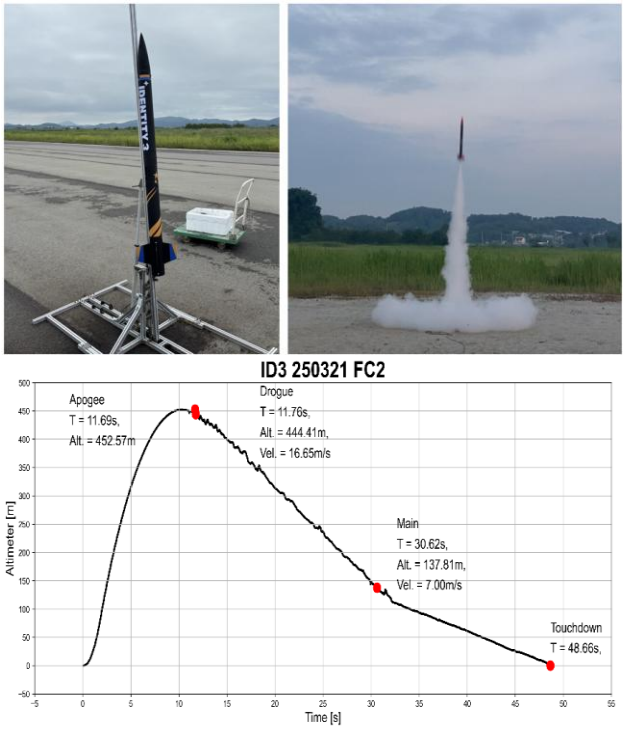
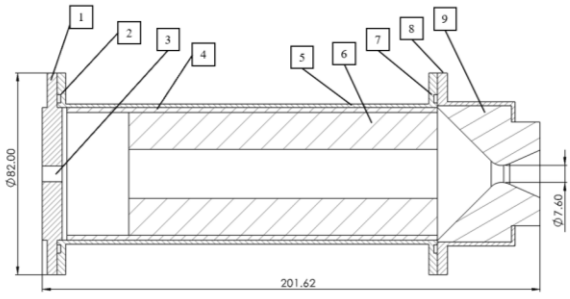


Fig. 13. Identity 3 Operation Result

Within this program, the propulsion team designed an I-class solid motor based on KNSB conducted more than 15 static fire tests for design validation and flight operations. Identity 3 static tests continue to be run as part of the team′s newcomer training for static-fire procedures and data processing using the integrated framework.

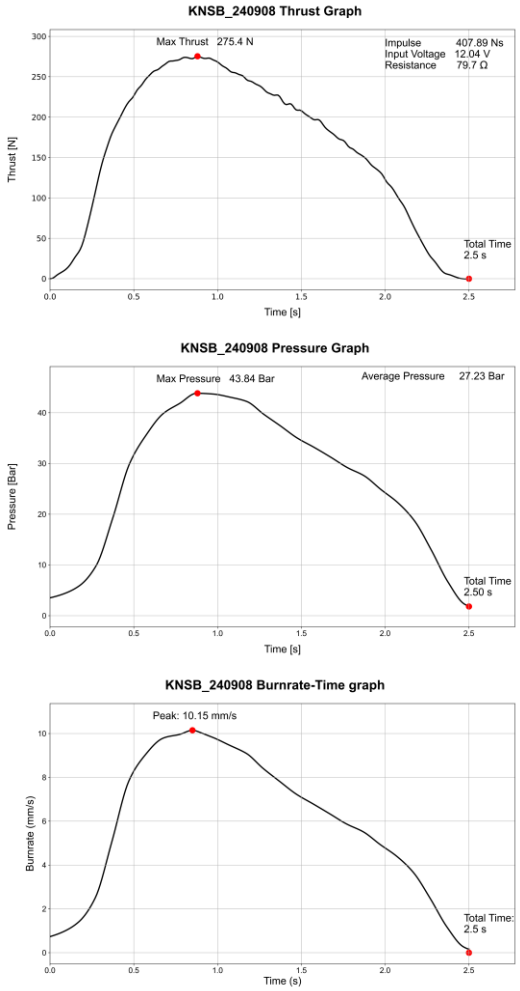


1	Aluminum Bulkhead	6	400g KNSB Bates Grain
2	Bulkhead O-ring	7	Nozzle O-ring
3	Hole for Pressure Transducer Fitting	8	Nozzle Mount
4	Grain Paper Tube	9	Graphite Nozzle
5	H-type Flange Chamber		

Fig. 13. Design of Identity 3 Motor

- I-class solid motor burning a single-grain, hollow-cylindrical KNSB (400 g) propellant.
- Aft-end ignition using a rear igniter co-developed with the Identity-3 motor.
- Chamber formed by symmetric H-shaped flanges, a configuration that later informed multiple Hanaro propulsion architectures.
- Design pressure of 60bar with an approximate safety factor of 2.5.
- An O-ring placed at each fore and rear flange.

Fig. 14. Static Firing of Identity 3 Motor



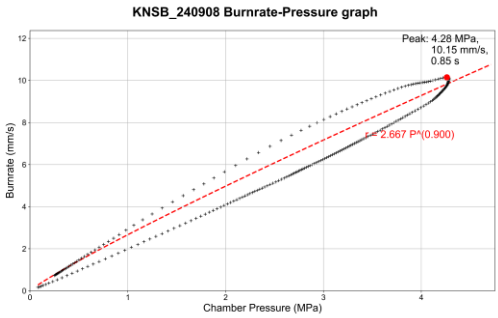


Fig. 15. Static Firing Data Analysis by HANARO SFT

After going through several static fire testing, we established basic characteristics for the Identity-3 motor and manufactured KNSB propellant. These foundation data were subsequently used to inform the design of other KNSB motors. While a hollow-cylindrical grain with inhibited outer surfaces typically exhibits a progressive time–thrust profile (rising gradually to a peak before a sharp drop) we consistently observed a regressive behaviour: an abrupt rise to peak thrust followed by a gradual decline. We attribute this to localized burning on nominally inhibited surfaces. Because the trend was repeatable across tests, we were able to incorporate the observed behaviour directly into operational planning. Burn-rate analysis indicated a high pressure-exponent with good repeatability across tests. We infer that the motor's relatively short length makes the regression rate particularly sensitive to small chamber-pressure variations

5.2 KHAOS: A M-class Sounding rocket for 25 IREC Preparation

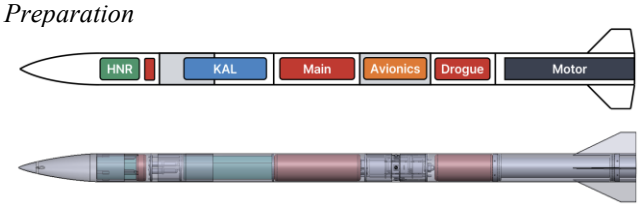


Fig. 15. KHAOS Subsystem Overview

KHAOS is a 2km-sounding rocket that was launched once in February as a test flight for the 2025 IREC and to verify an industrial payload in collaboration with Korean Air. Because transporting custom motors is not permitted, HARANG 2 was launched at IREC with a COTS motor while keeping all subsystems identical to those of KHAOS. The successful launch and recovery of KHAOS, including the successful ejection of HNR payload, validated each subsystem performance and materially contributed to the IREC mission outcome

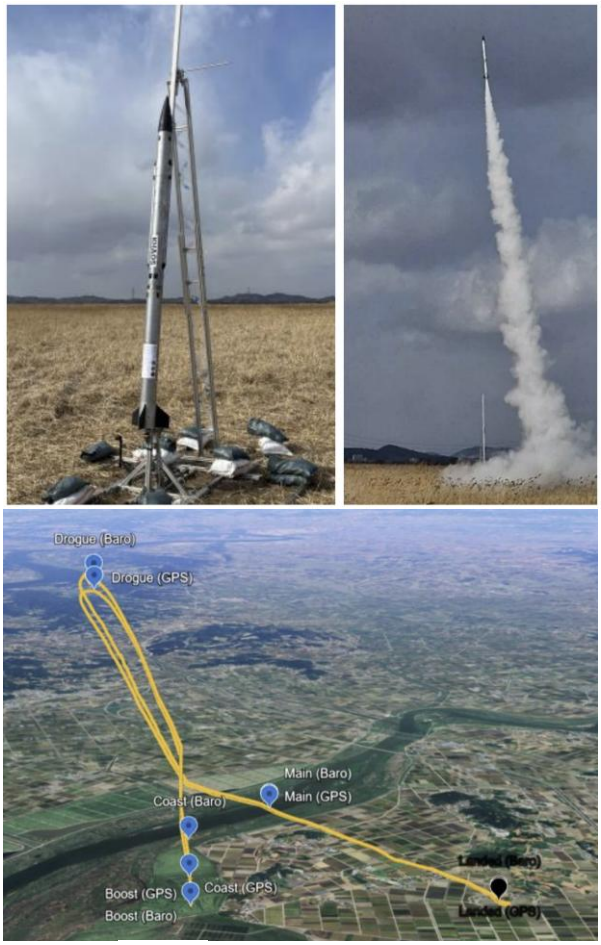
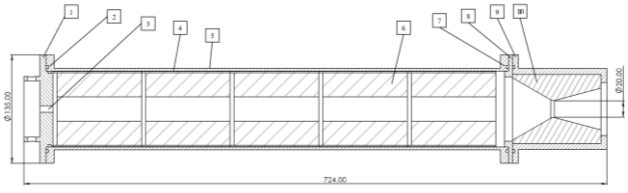


Fig. 16. KHAOS Operation Result

The propulsion team initially planned to develop an AN-based metalized solid propellant for KHAOS, but schedule delays led us to design an M-class KNSB motor instead. After five static fire testing, the motor was validated, and sufficient data were acquired for operations. Although the KHAOS propulsion system has not been operated after IREC, the experience of fielding a high-thrust motor substantially advanced our propulsion system testing and data-analysis framework.

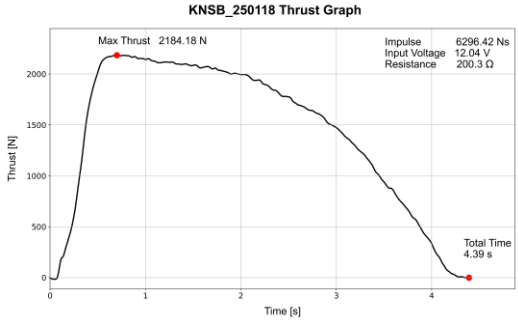


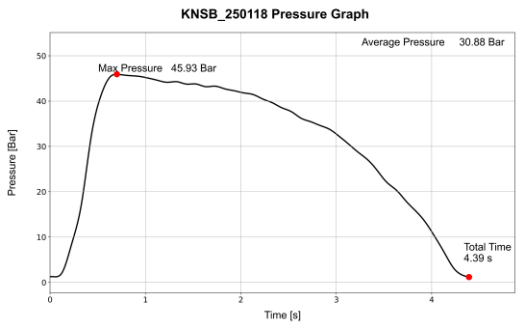
1	Aluminum Bulkhead	6	5-Segments of 1kg KNSB Grain
2	Bulkhead O-rings	7	Nozzle O-rings
3	Hole for Pressure Transducer Fitting	8	Nozzle Holder
4	Grain Paper Tube + Thermal Grease	9	Nozzle Mount
5	H-type Flange Chamber	10	Graphite Nozzle

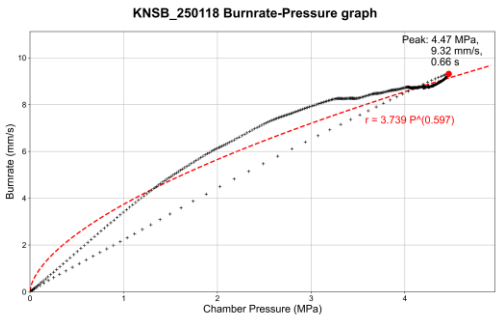
Fig. 17. Design of KHAOS Motor

- 5kg of KNSB separated into five hollow-cylindrical grains.
- Chamber formed by symmetrical H-shaped flanges, sealed by two O-rings at each end.
- Design pressure of 60bar with as safety factor of 2.15, the intentional failure mode is bulkhead bolt fracture to localize damage under over-pressure.
- Inner chamber wall is coated with thermal grease, with epoxy-bonded insulation segments inserted during motor assembly.

Fig. 18. Static Firing of KHAOS Motor







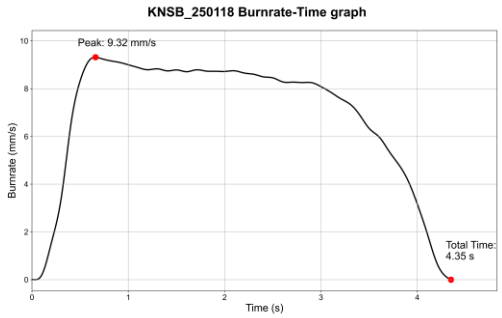


Fig. 19. Static Firing Data Analysis by HANARO SFT

Leveraging the design requirements of the thrust-measurement system hardware, we operated a test stand capable of accommodating multiple motor sizes; accordingly, propulsion verification for KHAOS was conducted on the same test-stand model used for Identity-3 after a simple load-cell swap (2.5kN to 5.0kN). Post analysis data indicates a nearly constant burn rate and a stable pressure exponent below 0.6.

However, after the first three tests, a bulge was observed on the chamber, prompting fabrication of a new unit. During the fourth firing, localized thermal-protection shortcomings led to softening of the outer case, leading to a gas leak.

Fig. 20. Failure of KHAOS Motor During Static Firing

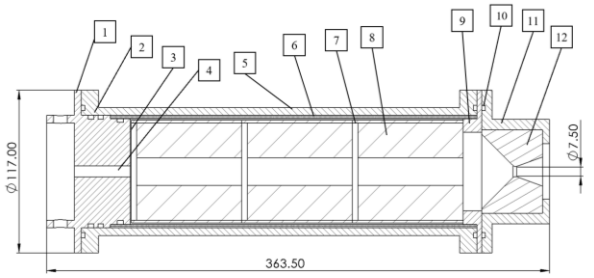
For the fifth firing, we bonded each segment with epoxy and inhibited the inner chamber surface with grease, after which the test succeeded, and the motor was cleared for flight use. Because designing a motor with a large KNSB mass was atypical for the Hanaro propulsion team, thermal protection received less emphasis than warranted. We therefore consolidated the operational outcomes and lessons learned into internal documentation to inform future designs and procedures.

5.3 Static Fire Testing of an Ammonium Nitrate-based propellent

To overcome the specific-impulse limitations of KNSB while improving manufacturability for higher-thrust applications, Hanaro is developing a metalized ammonium nitrate (AN) based solid propellant.

Fig. 21. AN-based solid propellant “MALANG”

Because most metalized composite propellants rely on ammonium perchlorate (AP), which is not obtainable in Korea, we explored multiple AN-based formulations to characterize their combustion behaviour through static firings. We used a small epoxy fraction as the binder and magnesium and aluminium as metallic additives. Under the program name ″MALANG,″ we conducted approximately ten static firings to screen candidate mixes and extract key combustion characteristics.



1	Aluminum Bulkhead	7	Grain Paper Tube
2	Bulkhead O-rings	8	3 Segments of “MALANG” Grain
3	Insulator Disk	9	Nozzle Holder
4	Hole for Pressure Transducer Fitting	10	Nozzle O-rings
5	H-type Flange Chamber	11	Nozzle Mount
6	Phenolic Tube	12	Single-Use Graphite Nozzle

Fig. 21. Design of MALANG Testing Motor

- 3 hollow cylindrical ″MALANG″ grains are mounted
- To ensure safe testing under unknown propellant characteristics, safety factor of the chamber is designed to be larger than 4.
- Insulated by phenolic tube and disk to prevent direct contact with aluminium chamber.
- Graphite nozzle replaced after single test, due to metallic residue after combustion.

Fig. 22. Static Firing of MALANG Test Motor

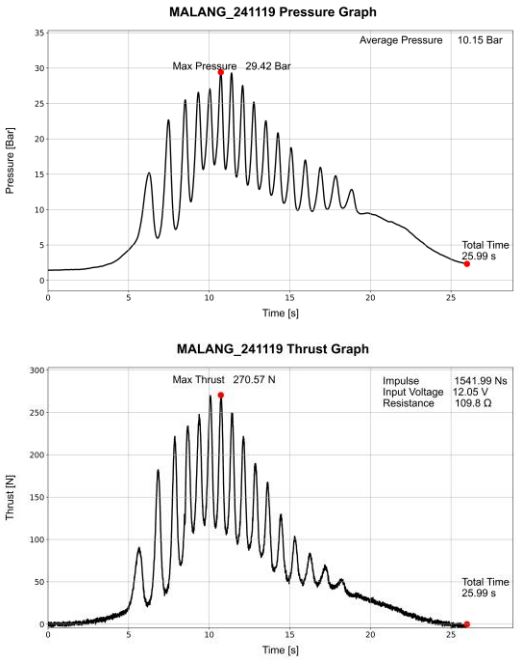


Fig. 23. Static Firing Data Analysis by HANARO SFT

We varied the nozzle throat diameter and the epoxy binder fraction across multiple static firings, but no meaningful performance improvements were observed. As shown in the plots, the thrust and chamber-pressure traces exhibited pronounced oscillations, and the burn durations were excessively long. During testing, nozzle choking led to three over-pressure events, which in turn prompted reinforcements to the test-stand design. Post-test data analysis indicated that the evaluated formulation is unsuitable for use as a flight propellant. We have therefore documented the procedures and datasets for future reference and restarted the program by replacing the binder with HTPB, beginning again at ambient-pressure combustion testing.

6. Software Engineering Practices for HANARO SFT

HANARO SFT (Static-Fire Toolkit) is an open-source, CLI-first toolkit that standardizes the processing of thrust/pressure data from solid rocket motor static-fire tests conducted by amateur and research teams; it is designed for integration into diverse testing workflows while maintaining scientific rigor and reproducibility. The public codebase is hosted on GitHub and distributed on the Python Package Index (PyPI) [14].

6.1 Design Philosophy and Distribution

We focused on reproducibility, installability, and maintainability across heterogeneous lab machines. Packaging for PyPI makes the tool discoverable, installable via pip, and classifiable with Trove classifiers while deferring hard runtime constraints to requires-python (as recommended by the Packaging User Guide). Releases are produced by CI on signed or versioned tags, then published to PyPI via Trusted Publishing (OIDC), reducing secrets management risk.

6.1.1. Package Distribution via PyPI

In order to ensure both reproducibility and accessibility of the HANARO SFT software, the Python Package Index (PyPI) was selected as the primary distribution channel. PyPI represents the standard repository within the Python ecosystem, enabling uniform installation and dependency resolution across heterogeneous computing environments. Its adoption facilitates not only straightforward deployment through the pip installer but also standardized metadata registration, versioning, and classifier usage in accordance with Python Enhancement Proposals (PEPs) on packaging and distribution (e.g., PEP 621 and PEP 517/518) [15,16,17,18].

By aligning with this infrastructure, the project ensures that researchers and student teams can install the toolkit without recourse to custom build procedures, while also benefiting from automatic integration into continuous integration workflows. Furthermore, the use of PyPI's trusted publishing framework minimizes the risks associated with credential management during release, thereby reinforcing the long-term sustainability and reproducibility of the software as a scholarly artifact.

6.1.2. Standards-compliant project structure

We adopt PEP 621 to declare core metadata in `pyproject.toml`, and PEP 517/518 to define a build-system interface and its build-time requirements [15,16,17,18]. This keeps metadata tool-agnostic and enables modern builders while remaining compatible with the prevailing ecosystem (e.g., setuptools as a backend). Command-line entry points use the [project.scripts] table, which maps the `sft` console command to our CLI.

The repository structure employs a standard `src/` layout with the main package under `src/static_fire_toolkit/`, ensuring compatibility with contemporary Python development tools and installation mechanisms.

The codebase implements a modular pipeline architecture where thrust processing, pressure analysis, and burn rate estimation operate as independent stages with well-defined interfaces. This design enables flexible workflow customization while maintaining data provenance through intermediate file outputs. Each processing stage validates input data integrity and logs processing parameters for reproducibility.

6.1.3. Versioning Policy

The project adheres to Semantic Versioning (SemVer), wherein MAJOR, MINOR, and PATCH numbers communicate compatibility guarantees to downstream users [19,20,21]. This ensures that breaking API changes are explicitly signaled, while incremental improvements and bug fixes can be adopted without risk. At the same time, release identifiers comply with PEP 440, Python′s canonical versioning specification, which guarantees compatibility with packaging tools and installers [22]. To prevent divergence between metadata and runtime, the toolkit dynamically reads its `__version__` from installed distribution metadata via `importlib.metadata`. This design ensures a single source of truth and was reinforced in the v1.0.1 release, which corrected inconsistencies between declared and runtime versions [21].

6.2 Engineering Considerations and Code Quality Assurance

Deployment of production-ready quality software requires systematic attention to reliability, maintainability, and user experience beyond core algorithmic functionality. Our implementation incorporates multiple layers of quality assurance designed to meet research-grade requirements for reliability, reproducibility, and accuracy.

6.2.1. Continuous Integration and Delivery

Our GitHub Actions pipeline include automated testing, strict code quality enforcement through ruff for linting and formatting, and auto-publishing on tag.

(1) Matrix Testing

Matrix testing across supported CPython versions (3.10 − 3.13) and two dependency strata: (i) latest releases and (ii) a ″minimum constraints″ set. This ensures APIs we rely on are available throughout the stated support window and guards against accidental tightening of version specifiers. Dependency specifiers follow PEP 508 and, transitively, PEP 440 [22,23].

(2) Linting and Formating

We unify linting and formatting under Ruff to reduce developer cognitive load and CI time. Ruff covers pycodestyle/pyflakes/pyupgrade/pydocstyle formatting rules in one tool, which is configured via pyproject.toml, and runs quickly enough to be enforced on every commit and/or PR [24].

(3) Trusted Publishing to PyPI

Trusted Publishing to PyPI via `pypa/gh-action-pypi-publish` GitHub Actions workflow with `id-token: write` [25], triggered only when a signed/annotated tag on the main branch is pushed. Optional environment protection (required reviewers / wait timers) can gate the publish job for additional control.

On successful publish, a GitHub Release is created and assets (sdist/wheel, SHA256 checksums) are attached using a dedicated action.

6.2.2. Refactoring for Generalization and Maintainability

During the transition from an internally used tool to a publicly released software package, substantial refactoring was undertaken to enhance generality, maintainability, and usability. Previous closed-source iterations of the code contained hardcoded parameters tied to specific data acquisition (DAQ) systems and data formats, which severely limited portability across different experimental setups.

In preparation for the v1.0.0 major release, these embedded constants and column index assumptions were systematically removed and replaced with user-configurable parameters. Configuration options were consolidated in dedicated files (`global_config.py` and `config.xlsx`)[14,21], allowing users to specify CSV delimiters, header row indices, load cell sensitivities, and other experiment-specific parameters. This design decision broadens applicability beyond the originating team, ensuring usability for both amateur and academic rocketry groups with various hardware environments.

At the same time, the codebase underwent a round of variable renaming and clarification. Several legacy variables with ambiguous or team-specific shorthand names were replaced with more descriptive identifiers. This improved readability and reduced the barrier to entry for new contributors, while also aligning with best practices for scientific software maintainability [14,21,26].

These refactoring steps not only improved software robustness but also transformed HANARO SFT from a narrowly scoped internal utility into a reusable research-grade library. This shift represents a deliberate engineering decision to prioritize long-term sustainability and cross-team collaboration, consistent with the broader goals of open-source scientific infrastructure.

6.2.3. Error Handling, Logging, and Diagnostic Support

Robust error handling and diagnostic support are essential for ensuring the reliability of production-ready software. In HANARO SFT, exception handling constructs are systematically applied to prevent uncontrolled termination and to provide informative error messages when unexpected conditions arise. This approach allows users to identify configuration or data format issues without compromising the reproducibility of the analysis workflow.

To further enhance traceability, the software integrates a logging subsystem that records detailed execution traces. Logging is applied not only to critical failures but also to intermediate processing steps, parameter values, and file access events. Such comprehensive logs serve a dual purpose: they provide end-users with transparency regarding the internal operation of the toolkit, and they supply developers with valuable feedback when diagnosing reported issues.

In addition, a dedicated diagnostic interface is offered through the `sft info` command. When executed, this command outputs the runtime environment information, including the execution path, Python interpreter version, versions of key dependencies, current values of global configuration parameters, and the presence or absence of the essential per-experiment configuration file (config.xlsx). By consolidating this information in a standardized format, HANARO SFT significantly reduces the effort required to reproduce user-reported issues, thereby facilitating efficient debugging and collaborative problem resolution.

Collectively, these measures establish a software environment in which both end-user reliability and developer maintainability are supported, aligning the toolkit with best practices for open-source scientific software engineering.

6.2.4. Trunk-based Development Workflow

To manage contributions efficiently within a small team primarily focused on hardware rather than software, we adopt trunk-based development. Short-lived feature branches fork directly from `main`, integrate early and often, and avoid long-lived divergence. This strategy minimizes merge conflicts and reduces the integration surface, aligning with best practices for small research-driven teams, where minimizing merge overhead is critical to maintaining velocity.

The `main` branch is protected with required status checks, enforced linear history, and mandatory reviews. In addition, tag protection rules are applied to ensure that release tags cannot be created, deleted, or moved without explicit authorization. Together, these measures maintain repository integrity and support reproducible, high-confidence releases.

6.3 Accessibility, Documentation, and Community Engagement

Ensuring accessibility requires attention not only to technical reliability but also to documentation, community structures, and long-term sustainability. For HANARO SFT, documentation is deliberately consolidated in the GitHub repository README to reduce the entry barrier for student and amateur teams. The README provides installation instructions, CLI usage guides, and worked examples that illustrate the complete analysis pipeline from raw data to processed outputs [14]. Release notes follow the Keep a Changelog convention and are mirrored into both annotated tags and GitHub Releases, ensuring consistent communication of changes across distribution channels [20,21].

To support scholarly attribution, Zenodo is integrated with the GitHub repository, enabling persistent DOI assignment for each release. This provides a formal citation mechanism, thereby aligning the software with academic publication standards and facilitating reproducibility in subsequent research.

Community engagement will be fostered through structured GitHub issue templates and pull request guidelines, supported by a code of conduct that establish clear norms of professional interaction. Although the primary interface language is English, the software accommodates international collaboration by supporting SI unit system and configurable output formatting, reflecting the diverse measurement practices of the global rocketry community.

Sustainability is further reinforced through the adoption of the MIT License, which guarantees permissive reuse and long-term availability independent of institutional context. A modular architecture and diagnostic tools (e.g., `sft info`) lower the cost of maintenance by contributors, ensuring HANARO SFT's durability as a component of the broader open-source ecosystem for amateur and academic rocketry.

7. Discussion

The present framework targets the end-to-end path from static-fire testing of solid motors to the use of reliable data in mission operations, and its reproducibility has been demonstrated by publishing procedures and results from Hanaro's repeated field use with more than 30 static fire tests and 11 launch missions. To extend the approach to propulsion systems beyond sounding rockets, modest adaptations such as adding specific sensors or post-processing modules may be required; however, we expect these extensions can be incorporated without altering the core architecture. We also view the open release of this system as establishing a baseline for low-cost propulsion test infrastructure. Building on this foundation, we anticipate evolving the framework into a more general platform for testing and evaluation of more advanced propulsion systems.

This work began as a response to practical constraints faced by a student team and matured into an integrated framework that we used to design, test, and operate solid-motor systems for real missions. After completing those missions successfully, we chose to publish the hardware design, analysis software, and

representative datasets so that other engineers can reproduce, adapt, and improve the approach. We see this as the first step in a broader movement: tackling shared problems together, documenting what works, and lowering the entry barrier for future teams.

While solid propulsion is no longer the central driver of the space industry and sounding rockets are sometimes viewed as a hobby activity, the discipline of building, testing, and operating these systems provides concrete engineering insight and a first major leap for new practitioners. The framework and the lessons summarized here were shaped by repeated field operations; by releasing them openly, our goal is to provide a foundation others can stand on rather than asking each team to start from zero.

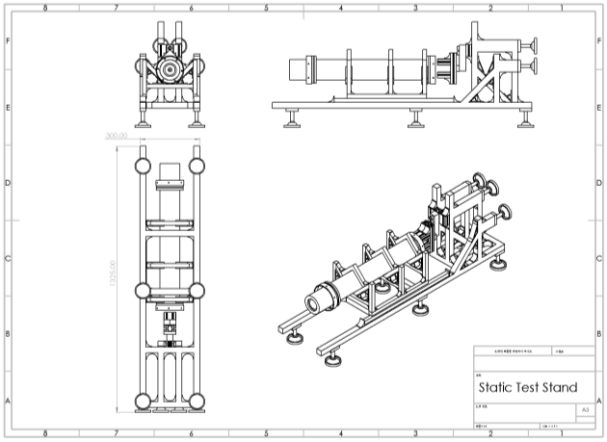
We hope readers will use, modify, and extend the framework to reach higher apogees in their own projects, and that shared infrastructure with open documentation will shift the community from isolated competition toward cooperative progress.

Acknowledgements

The authors gratefully acknowledge the continuous guidance and encouragement of Prof. Bok Jik Lee, Seoul National University. We would also like to thank the members of the SNU Rocket Team Hanaro for their dedicated contributions to the design, manufacturing, static firing, and data analysis of the solid rocket motors described in this paper.

The collaboration with Korean Air for the industrial payload sub-mission of HARANG 2 is also acknowledged. Finally, the authors thank the broader amateur rocketry community for their continued support in enabling student-driven aerospace research.

Appendix A: Static Test Stand Component Details



No.	Components	Quantity
1	Leveling Foot	4(10*)
2	30 mm 45° Bracket	26
3	30 mm 90° Bracket	54
4	20 mm 90° Bracket	28
5	30 mm 135° Bracket	4
6	Foot Base 3060	5(11*)
7	20×20×234 mm Aluminum Profile	3
7	20×20×234 mm Aluminum Profile	3
8	30×30×1300 mm Aluminum Profile	2
9	30×30×240 mm Aluminum Profile	9
10	30×30×240 mm Aluminum Profile, 45° Cut	4
11	30×30×400 mm Aluminum Profile	2
12	20×20×400 mm Aluminum Profile	2
13	20×20×170 mm Aluminum Profile	6
14	20×20×165.46 mm Aluminum Profile, 45° Cut	4(6**)
15	30×60×90 mm Aluminum Profile	2
16	30×60×200 mm Aluminum Profile	2
17	30×120×60 mm Aluminum Profile	1
18	30×60×155 mm Aluminum Profile	1
19	30×65×60 mm Aluminum Profile	2
20	600 mm Linear Guide (Rail)	2
21	Linear Bearing Block	4
22	20×20×(*Motor Diameter*) mm Aluminum Profile	2(3**)
23	20×20×(*Motor Diameter* + 20) mm Aluminum Profile	2(3**)
24	60 x 60 mm Flat Bracket	4
25	3060 Aluminum Profile End Cap	2

* For optional lower levelling foot(6)
** 3 Mounting frames for larger motor

References

[1] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. J. Nature Methods 17(3) (2020) 261-272. DOI: 10.1038/s41592-019-0686-2.

[2] The SciPy community, scipy.signal.butter API Reference, 23 July 2025, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.signal.butter.html, (accessed 25.09.20).

[3] The SciPy community, scipy.signal.sosfiltfilt API Reference, 17 May 2025, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.signal.sosfiltfilt.html, (accessed 25.09.20).

[4] F. N. Fritsch, J. Butland, A method for constructing local monotone piecewise cubic interpolants, SIAM J. Sci. Comput. 5(2) (1984) 300-304. DOI: 10.1137/0905021.

[5] The SciPy community, scipy.interpolate.PchipInterpolator API Reference, 22 March 2025, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.interpolate.PchipInterpolator.html, (accessed 25.09.22).

[6] A. *Savitzky, M.J.E. Golay, Smoothing and Differentiation of Data by Simplified Least Squares Procedures, J. Analytical Chemistry. 36(8) (1964) 1627-39. DOI:* 10.1021/ac60214a047.

[7] The SciPy community, scipy.signal.savgol_filter API Reference, 29 July 2024, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.signal.savgol_filter.html, (accessed 25.09.22).

[8] C.R. Harris, K.J. Millman, S.J. van der Walt et al. *Array programming with NumPy*. J. Nature. 585 (2020) 357–362. DOI: 10.1038/s41586-020-2649-2.

[9] NumPy Developers, numpy.diff API Reference, 24 July 2024, https://numpy.org/doc/2.1/reference/generated/numpy.diff.html, (accessed 25.09.13).

[10] The SciPy community, scipy.ndimage.gaussian_filter1d API Reference, 26 July 2025, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.ndimage.gaussian_filter1d.html, (accessed 25.09.22).

[11] The SciPy community, scipy.integrate.simpson API Reference, 31 March 2025, https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simpson.html#scipy.integrate.simpson, (accessed 25.09.13).

[12] P. Vieille, Étude sur le mode de combustion des matières explosives, J. Mémorial des Poudres et Salpêtres. 6 (1893) 256-391.

[13] The SciPy community, scipy.optimize.curve_fit API Reference, 22 March 2025, https://docs.scipy.org/doc/scipy-1.16.2/reference/generated/scipy.optimize.curve_fit.html, (accessed 25.09.25).

[14] Y. Kim, J. Seo, J. Yun, **Static-Fire Toolkit,** 2025, https://github.com/snu-hanaro/static-fire-toolkit. DOI: 10.5281/zenodo.17218595

[15] The Python Packaging Authority (PyPA), pyproject.toml specification, 17 September 2025, https://packaging.python.org/en/latest/specifications/pyproject-toml/, (accessed 25.09.26).

[16] Python Enhancement Proposals (PEPs), PEP 621 – Storing project metadata in pyproject.toml, 31 October 2020, https://peps.python.org/pep-0621/, (accessed 25.09.26).

[17] Python Enhancement Proposals (PEPs), PEP 518 – Specifying Minimum Build System Requirements for Python Projects, 13 May 2016, https://peps.python.org/pep-0518/, (accessed 25.09.26).

[18] Python Enhancement Proposals (PEPs), PEP 517 – A build-system independent format for source trees, 11 September 2017, https://peps.python.org/pep-0517/, (accessed 25.09.26).

[19] T. Preston-Werner, Semantic Versioning 2.0.0, 19 December 2023, https://semver.org/#semantic-versioning-200, (accessed 25.09.18).

[20] O. Lacan, T. Fortune, Keep a Changelog version 1.1.0, 15 February 2019, https://keepachangelog.com/en/1.1.0/, (accessed 25.09.18).

[21] Y. Kim, Static-Fire Toolkit CHANGELOG, 19 September 2025, https://github.com/snu-hanaro/static-fire-toolkit/blob/main/docs/CHANGELOG.md.

[22] Python Enhancement Proposals (PEPs), PEP 440 – Version Identification and Dependency Specification, 22 August 2014, https://peps.python.org/pep-0440/, (accessed 25.09.27).

[23] Python Enhancement Proposals (PEPs), PEP 508 – Dependency specification for Python Software Packages, 16 November 2015, https://peps.python.org/pep-0508/, (accessed 25.09.27).

[24] Astral, Ruff Docs, 18 September 2025, https://docs.astral.sh/ruff/rules/, (accessed 25.09.20).

[25] The Python Packaging Authority (PyPA), Publishing package distribution releases using GitHub Actions CI/CD workflows, 17 September 2025, https://packaging.python.org/en/latest/guides/publishing-package-distribution-releases-using-github-actions-ci-cd-workflows/, (accessed 25.09.26).

[26] G. Wilson, D.A. Aruliah, C.T. Brown, N.P. Chue Hong, M. Davis, R.T. Guy, S.H.D. Haddock, K.D. Huff, I.M. Mitchell, M.D. Plumbley, B. Waugh, E.P. White, P. Wilson, Best Practices for Scientific Computing, J. PLoS Biol. 12(1) (2014) 1-7. DOI: 10.1371/journal.pbio.1001745.