Electro Thermal Lifetime Assessment for Inverter Semiconductors IGBTs and Diodes Documentation

Anirudh Katoch

13.10.2025

1 Introduction

The penetration of inverter based resources (IBRs) in modern electrical power systems has increased dramatically in recent years, primarily driven by the rapid growth of renewable energy technologies such as photovoltaic (PV) systems and wind turbines. These renewable sources interface with the grid through power electronic inverters, leading to a gradual decline in the share of conventional synchronous generators. Traditionally, synchronous generators have inherently provided both active and reactive power support, contributing significantly to voltage regulation and system stability. In contrast, many IBRs are typically operated at or near unity power factor and therefore do not inherently supply reactive power unless explicitly controlled to do so. Consequently, as the proportion of synchronous generation continues to decrease, the responsibility for reactive power support and voltage regulation is increasingly being transferred to inverters particularly grid-forming inverters which are designed to not only deliver active power but also actively regulate voltage and share reactive power with the grid. With the accelerating integration of IBRs, the role and reliability of these grid forming inverters have become critical to maintaining grid stability and performance [1].

However, when inverters are required to supply reactive power or deviate from unity power factor, their internal power semiconductor devices (IGBTs and diodes) are subjected to increased electrical and thermal stresses. For example, reactive currents can increase conduction losses, shift current distributions, and induce additional temperature cycling, which accelerate fatigue and degradation in the switches [2]. Because the lifetime of the inverter is often limited by the lifetime of its switching devices, it is critical to assess how inverter operating conditions in particular, the real power output and the associated power factor impact switch lifetime.

To address this need, this work presents a Python-based electro-thermal lifetime assessment model that estimates the lifetime of inverter semiconductor devices under varying electrical and thermal operating conditions. The framework takes as inputs the inverter's output power levels, power factor (i.e., reactive power demand), device parameters, and thermal boundary conditions along with other factors and evaluates the coupled electro-thermal stresses acting on the switching devices. In addition to these primary electro-thermal inputs, the model also incorporates secondary physical parameters that influence device degradation, including the wire-bond aspect ratio, die-attach material properties and thermal interface resistance. These factors are known to significantly affect thermo-mechanical fatigue mechanisms such as wire-bond lift-off, heel cracking and solder or sinter layer fatigue [2, 3].

Further in this document the python model will be explained. It is important to note that this document is dedicated solely to explaining the development and implementation of the electro-thermal lifetime assessment model. The primary objective here is to provide a comprehensive understanding of the underlying programming logic, data flow and algorithmic structure that enable the model to perform lifetime estimation of inverter semiconductor devices. Detailed simulation studies, validation results and performance analyses are beyond the scope of this document and will be presented separately in a dedicated research paper focusing on the model's experimental evaluation and application to real-world inverter mission profiles.

2 Methodology

The developed model is primarily based on two foundational works that is [2] and [3], with additional lifetime accumulation concepts adapted from [4]. Paper [3] introduces the empirical lifetime model derived from the LESIT power cycling experiments, which forms the mathematical core of the lifetime estimation framework. The model expresses the number of thermal cycles to failure, N_f, as a function of several stress-related parameters:

$$N_f = A \times (\Delta T_j)^{\alpha} \times (ar)^{\beta_1 \Delta T_j + \beta_0} \times \left[\frac{C + (t_{on})^{\gamma}}{C + 1} \right] \times \exp\left(\frac{E_a}{k_b \times \bar{T}_j} \right) \times f_d \tag{1}$$

Here:

- N_f - Number of thermal cycles to failure.
- ΔT_j - Junction temperature swing between the thermal cycle's maximum and minimum temperature, representing thermal fatigue severity.
- $\bar{T}_j$ - Mean junction temperature during thermal cycle.
- ar - Wire-bond aspect ratio, defined as the loop height to length ratio. This term captures mechanical reliability effects related to bond geometry.
- t_{on} - Pulse or thermal cycle duration, indicating how long the device remains under high stress during each cycle.
- $A, \alpha, \beta_0, \beta_1, C, \gamma, E_a, k_b, f_d$ - Empirical parameters obtained from LESIT experimental data for IGBT modules.

Equation (1) provides the number of cycles to failure for a semiconductor device under defined operating stresses.

Paper [2] provides the theoretical and computational framework for determining the electrical and thermal parameters required to evaluate (1). One basically needs the switching and conduction losses of both the IGBT and diode which are calculated as functions of the instantaneous current flowing through each device for a given inverter configuration. The model takes into account the inverter's operating power power factor, and a comprehensive range of device-specific parameters obtained directly from the respective IGBT and diode datasheets. These parameters are used to compute the instantaneous conduction and switching power losses, which collectively define the total device power dissipation under different loading and reactive power conditions. The resulting losses are subsequently applied to a Foster thermal impedance network to determine the transient junction temperature profiles. The implemented thermal network also incorporates the effects of the thermal interface materials, heat sink characteristics and cooling conditions, thereby accurately modeling the overall thermal path from junction to ambient. Using this combined electro-thermal model, both the mean junction temperature ($\bar{T}_j$) and the temperature swing (ΔT_j) are extracted along with (t_{on}), which are essential inputs to the subsequent lifetime estimation analysis.

Furthermore the cumulative lifetime of the device over a mission profile is then obtained using Miner's linear damage rule [4]:

$$LC = \sum_i \frac{n_i}{N_{f,i}} \tag{2}$$

where n_i represents the number of thermal cycles experienced at a particular stress condition i and $N_{f,i}$ is the number of cycles to failure predicted by (1) under that same condition. The device is considered to reach its end of life when $LC = 1$, indicating full life consumption.

To account for non-cyclic and time-dependent degradation effects such as diffusion, oxidation or corrosion in packaging materials, an additional Acceleration factor (AF) is introduced. The acceleration factor quantifies how much faster degradation occurs under elevated stress conditions compared to normal operating conditions [5]. In this work, the acceleration factor is incorporated into the cumulative lifetime estimation to account for long term calendric aging effects that are not directly related to cyclic thermal stress. The overall life consumption, initially calculated from cyclic electro-thermal loading using the LESIT–Miner model, is scaled by the acceleration factor as:

$$LC_{\text{total}} = AF \times LC_{\text{cyclic}} \tag{3}$$

where LC_{cyclic} represents the damage accumulated due to temperature cycling, and LC_{total} includes the additional contribution of time dependent aging mechanisms. In practice, this adjustment ensures that even under low power or idle conditions, slow degradation processes such as diffusion, intermetallic growth or corrosion continue to consume a fraction of the device lifetime.

Overall, the methodology combines the electro thermal modeling principles of [2] for temperature and loss estimation, the empirical LESIT lifetime model from [3] and the damage accumulation approach from [4]. This integrated framework enables quantitative prediction of IGBT and diode lifetimes under variable power factors, thermal cycling conditions, and long-term operating stresses, thereby allowing estimation of the overall inverter lifetime.

3 Model Architecture

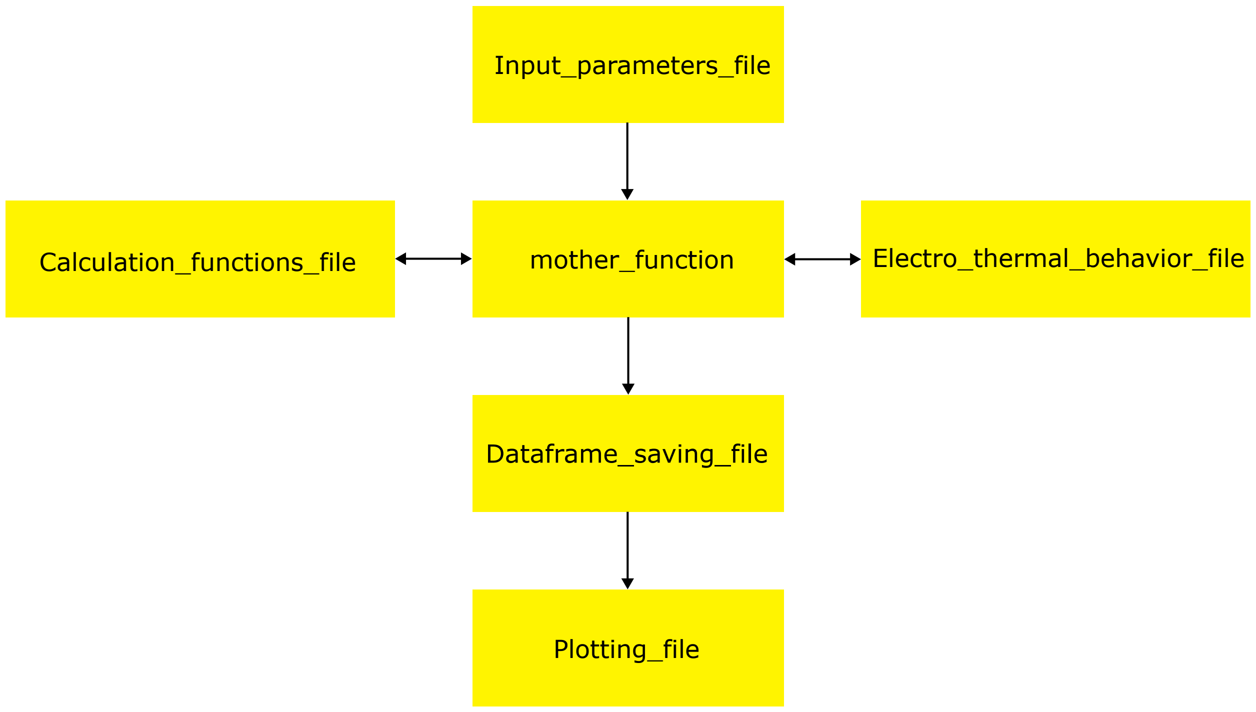

Figure 1: Model architecture.

The complete computational framework of the developed electro–thermal lifetime assessment model is illustrated in Fig. 1. The flowchart summarizes the interaction among the main Python scripts and the sequence of physical domains represented in the model. The process begins with the definition of the inverter, device, IGBT, Diode and thermal parameters and progresses through electrical loss computation, thermal modelling and lifetime estimation, before final visualization and data export. Each component of the flowchart corresponds directly to a functional module in the codebase.

- **Input_parameters_file.py** – This module defines all input variables for the analysis, including inverter operating conditions, device datasheet parameters and thermal parameters of the junction to ambient path. It acts as the central configuration file for the simulation.
- **Calculation_functions_file.py** – This file serves as the computational core of the entire framework, containing the majority of the analytical and numerical functions used throughout the model. It includes routines for calculating the switching and conduction losses of both IGBTs and diodes, as well as auxiliary functions that support subsequent thermal and lifetime analyses. In addition to electrical loss calculations, this module implements subroutines for evaluating RMS and average current values, determining thermal parameters, interpolating datasheet values and performing intermediate transformations required by the electro–thermal and lifetime models.
- **Electro_thermal_behavior_file.py** – Implements the thermal impedance network using the Foster RC model. It transforms the calculated power losses into junction temperature responses $T_j(t)$ and

extracts the mean junction temperature $\bar{T}_j$ and the temperature swing ΔT_j. The model accounts for the effect of the heat sink, thermal interface resistance and ambient temperature.

- **mother_function.py** – Acts as the main controller that integrates the electrical and thermal modules. It executes the lifetime calculation using the LESIT-based empirical model and Miner's rule for cumulative damage.
- **Dataframe_saving_file.py** – Stores all intermediate and final outputs such as temperature profiles, loss values and lifetime statistics into structured data frames. These are exported to the `Finished_results` directory for post-processing or reporting.
- **Plotting_file.py** – Handles visualization tasks including plots of switching and conduction losses, junction temperature variations, and the overall life-consumption evolution. These visualizations aid in validating the physical behavior of the model and provide insight into the most thermally and electrically stressed conditions.

Together, these modules form a sequential workflow that mirrors the physical process within an inverter that is electrical loading generates power losses, which cause thermal fluctuations, leading to device aging and lifetime reduction. This modular implementation ensures that individual components—such as the thermal model, lifetime equations or device libraries can be easily updated or replaced for different device characteristics without altering the overall structure.

4 Detailed Description of Python Modules

This chapter provides a comprehensive overview of each Python module that forms the foundation of the developed electro–thermal lifetime assessment framework. Each module in the repository has been designed with a specific purpose ranging from input parameter configuration and mission profile generation to electrical loss computation, thermal modelling, lifetime estimation, and result visualization. Understanding the role and operation of each file is essential for replicating, modifying or extending the model to new inverter topologies, semiconductor technologies or operating conditions. The following subsections describe these modules in detail, outlining their objectives, key functions, data flow and interaction with other components of the framework.

4.1 Input_parameters_file.py

This module initializes and stores all input parameters required by the model. It defines the operating conditions, mission profile, thermal and electrical device parameters, lifetime constants, and control flags for saving and plotting.

4.1.1 Lifetime model parameters

Table 1 summarizes all constants and coefficients derived from the *LESIT (Lifetime Evaluation of Semiconductor Interconnect Technologies)* model, which correlates junction temperature variations with the expected number of power cycles to failure [3]. The lifetime model parameters define the empirical constants used to evaluate the thermal fatigue and degradation behavior of the semiconductor devices.

Table 1: Lifetime model coefficients and constants.

Variable	Type	Unit	Description
`A`	float	–	Pre-exponential factor in the LESIT lifetime model. Determines the baseline number of cycles to failure.
`alpha`	float	–	Exponent on the junction temperature swing (ΔT_j). A negative value indicates that higher temperature swings reduce lifetime.
`beta1`	float	–	Exponent on mean junction temperature ($\bar{T}_j$), captures the temperature dependence of degradation rate.
`beta0`	float	–	Coefficient representing the sensitivity to the wire-bond aspect ratio (ar).
`C`	float	–	Dimensionless constant modifying the influence of heating time or duty cycle.
`gamma`	float	–	Exponent related to the heating duration (t_{on}). Negative values imply that shorter heating times extend device life.
`fd`	float	–	Scaling factor for the Diode.
`fI`	float	–	Scaling factor for the IGBT. Typically set to 1 for reference normalization.
`Ea`	float	J	Activation energy of the thermally driven degradation mechanism. Defines the temperature sensitivity of aging.
`k_b`	float	J/K	Boltzmann constant.
`ar`	float	–	Wire-bond aspect ratio, which influences the mechanical fatigue behavior of the device interconnects.
`t_cycle _heat _my_value`	float	s	This is thermal cycle duration which is t_on from Equation 1. Here the users defines its own value of thermal cycle duration. It should be noted that in the Python model `t_on` is effective turn on duration.

4.1.2 Thermal Model Parameters

The thermal subsystem of the model represents heat flow from the semiconductor junctions to the ambient environment through multiple material layers that is package, thermal interface material (TIM), and heat sink. Each layer is modeled as a Foster RC network, consisting of one or more parallel thermal resistances (R_{th}) and capacitances (with time constants $\tau = R_{\text{th}}C_{\text{th}}$). The Foster model captures transient thermal behavior and is used to compute the junction temperature response $T_j(t)$ when subjected to time-varying power losses. The Python implementation defines these parameters for each thermal path paste, sink, IGBT, and diode based on datasheet or manufacturer information. Table 2 summarizes all thermal variables, their data types, units, and physical meaning.

Table 2: Thermal model parameters

Variable	Type	Unit	Description
`r_paste`	ndarray	K/W	Thermal resistance of the case-to-sink thermal interface.
`tau_paste`	ndarray	s	Associated thermal time constant of the interface layer, representing its transient thermal response.
`r_sink`	ndarray	K/W	Thermal resistance of the heat sink (sink-to-ambient path)
`tau_sink`	ndarray	s	Thermal time constants of the sink-to-ambient path, capturing multi-time-constant behavior due to heat spreading and convection.
`r_I`	ndarray	K/W	Foster RC resistance coefficients for the IGBT junction-to-case thermal path.
`tau_I`	ndarray	s	Corresponding thermal time constants for the IGBT junction-to-case path.
`r_D`	ndarray	K/W	Foster RC resistance coefficients for the diode junction-to-case thermal path.
`tau_D`	ndarray	s	Time constants for the diode junction-to-case path.
`Tamb`	float	s	Ambient temperature (It remains constant throughout the simulation).
`N_FOSTER`	int	–	Fixed number of RC branches to which all Foster arrays are resized for numerical consistency.

The Foster model outputs the thermal impedance response $Z_{\text{th}}(t)$, which is later convolved with the instantaneous power loss waveform to obtain the junction temperature $T_j(t)$. To improve computational efficiency, the discrete-time decay factors for each thermal RC branch are precomputed using the sampling time step Δt:

$$\alpha = \exp\left(-\frac{\Delta t}{\tau}\right) \tag{4}$$

This is implemented as a set of Python `@property` functions for each thermal element, as summarized in Table 3. These parameters together form the basis of the transient thermal model used by the function `Electro_thermal_behavior_file.py` to determine the junction temperature evolution of the IGBT and diode under any given power-loss profile.

Table 3: Precomputed thermal decay factors.

Variable	Type	Unit	Description
`alpha_I`	property	–	Discrete-time exponential decay factor for each RC branch of the IGBT junction-to-case network.
`alpha_D`	property	–	Equivalent decay factor for the diode junction-to-case network.
`alpha_p`	property	–	Decay factor for the thermal interface layer (paste).
`alpha_s`	property	–	Decay factor for the heat sink thermal branches.

4.1.3 IGBT and Diode related parameters and constraints

This section defines the key electrical and operational limits for the semiconductor devices (IGBT and diode) used within the electro–thermal lifetime framework. These parameters ensure that the simulation remains within the safe operating area of the power module, thereby preventing numerical instabilities or non-physical results. The defined constraints originate from the device datasheets and manufacturer specifications, and they represent the maximum permissible electrical, thermal, and temporal operating limits for the switching components.

Switching Device Limits

The switching limit parameters specify the maximum allowable voltage, current, and temperature for both the IGBT and diode, as summarized in Table 4. These values are used by the simulation routines to flag or terminate any operating conditions exceeding safe limits. Additionally, an overshoot margin is applied to account for transient spikes during switching events.

Table 4: Switching device maximum operating limits.

Variable	**Type**	**Unit**	**Description**
`max_V_CE`	float	V	Maximum collector–emitter blocking voltage for the IGBT.
`max_IGBT_RMS_Current`	float	A	Maximum RMS current that can be continuously conducted by the IGBT.
`max_IGBT_peak_Current`	float	A	Peak allowable IGBT current during short transients or pulse operation.
`max_Diode_RMS_Current`	float	A	Maximum RMS current for continuous diode conduction.
`max_Diode_peak_Current`	float	A	Maximum allowable peak current through the diode during transient conduction or recovery.
`max_IGBT_temperature`	float	K	Absolute maximum junction temperature of the IGBT. Corresponds to approximately 175°C.
`max_Diode_temperature`	float	K	Maximum junction temperature for the diode.
`overshoot_margin`	float	–	Safety margin applied to account for voltage and current overshoots during dynamic operation.

Lifetime Constraints

The maximum allowable service life of the power devices is defined by the parameters `IGBT_max_lifetime` and `Diode_max_lifetime`, both set to 30 years. These values establish upper bounds for mission profile simulations and are used as reference points when estimating aging and remaining useful life. This is to be noted that these values come from the user and not from the datasheet. These parameters also work as a safeguard cause in some cases the lifetime can overshoot.

Table 5: Defined lifetime constraints for semiconductor devices.

Variable	**Type**	**Unit**	**Description**
`IGBT_max_lifetime`	float	years	Maximum design lifetime for the IGBT. Used as reference for mission-cycle comparisons.
`Diode_max_lifetime`	float	years	Equivalent maximum lifetime for the diode.

Switching Loss Parameters

Switching losses occur during device turn-on and turn-off transitions due to simultaneous voltage and current overlap. The model defines the switching frequency and effective transition times based on the manufacturer's datasheet. These parameters are temperature dependent but for the simplicity of this model the values are taken at constant temperature. The relevant parameters are shown in Table 6

Table 6: Switching loss parameters for IGBT and diode.

Variable	**Type**	**Unit**	**Description**
			IGBT
`f_sw`	float	Hz	Inverter switching frequency defining the number of commutations per second.
`t_on`	float	s	Effective turn-on duration, comprising both delay (t_d) and rise time (t_r). This value should not be confused with t_on defined in Equation 1. The t_on defined in Equation 1 have a different variable name in Python model which is `t_cycle_heat_I` for IGBT and `t_cycle_heat_D` for Diode.
`t_off`	float	s	Effective turn-off duration, including delay (t_d) and fall time (t_f).
			Diode
`I_ref`	float	A	Reference test current for reverse recovery measurement.
`V_ref`	float	V	Reference test voltage used for reverse recovery characterization.
`Err_D`	float	J	Reverse recovery energy per switching event under specified test conditions.

Conduction Loss Parameters

The conduction losses model is based on a simplified linear representation of the I–V characteristics of the semiconductor devices. Each device is described by a threshold (knee) voltage and an effective on-state resistance, as given in Table 7. The parameter values are current and temperature dependent but for the simplicity constant temperature and current is assumed.

Table 7: Conduction loss parameters for IGBT and diode.

Variable	**Type**	**Unit**	**Description**
IGBT			
`R_IGBT`	float	Ω	Effective on-state resistance.
`V_0_IGBT`	float	V	Knee voltage representing threshold for IGBT conduction onset.
Diode			
`R_D`	float	Ω	Dynamic on-resistance representing slope of the forward voltage–current curve.
`V_0_D`	float	V	Forward knee voltage of the diode.

4.1.4 Inverter input configuration parameters

This subsection defines the configuration settings related to the inverter topology, modulation method and thermal coupling assumptions between semiconductor devices. These user-defined parameters control how the electro–thermal model initializes and interprets the overall inverter structure. Each parameter is validated at initialization to ensure that it adheres to the predefined set of allowable options, thus preventing invalid configurations during simulation.

Table 8: Inverter configuration and input parameters.

Variable	**Type**	**Unit**	**Description**
`thermal_states`	str	–	Defines the thermal coupling between the IGBT and diode. If set to `"shared"`, both devices share the same thermal interface (paste and heat sink). If set to `"separated"`, each device has its own distinct thermal path.
`inverter_phases`	int	–	Specifies the number of inverter phases. Can be either `1` (single-phase) or `3` (three-phase).
`modulation_ scheme`	str	–	Determines the pulse-width modulation (PWM) method for controlling the inverter. Acceptable options are: `"spwm"` (Sinusoidal PWM) and `"svm"` (Space Vector Modulation). Applicable only when `inverter_phases = 3`.
`single_phase_ inverter_topology`	str	–	Defines the inverter topology for single-phase operation. The two valid options are `"half"` (half-bridge) and `"full"` (full-bridge). This parameter is ignored when `inverter_phases = 3`.
`design_control`	str	–	Specifies the level of design abstraction used in the sizing process. Options include `"inverter"` (system-level design, automatically selecting parallel devices to meet power demand) and `"switch"` (individual device-level design with direct power input).
`N_parallel`	int	–	Number of semiconductor devices connected in parallel per switching leg when operating in `design_control = "inverter"` mode. If None then the model defines `N_parallel` automatically based on safety limits.
`overshoot _margin_inverter`	float	–	Defines the overshoot margin factor for inverter-level voltage or current transients. This value adjusts allowable dynamic headroom during switching transitions.

4.1.5 Electrical flow and power variables

The parameters as shown in Table 9 defines the key electrical quantities governing the inverter's steady state operation and interaction with the AC grid.

Table 9: Electrical flow and power variables.

Variable	**Type**	**Unit**	**Description**
`P`	ndarray	W	Array containing the inverter's output active power values for each operating point or mission cycle segment.
`pf`	ndarray	–	Power factor corresponding to each power operating point. Defines the ratio between active and apparent power.
`Q`	ndarray	var	Array of reactive power values computed or provided for each operating condition.
`Vs`	ndarray	V	Inverter phase RMS voltage on the AC side.
`V_dc`	ndarray	V	DC-link voltage supplied to the inverter.
`f`	float	Hz	Fundamental frequency of the AC grid connected to the inverter.
`M`	float	–	Modulation index defining the amplitude ratio between the inverter's reference voltage and the DC-link voltage.
`omega`	property (float)	rad/s	Angular frequency of the grid. This property ensures automatic consistency if `f` is modified.

Note: It should be noted that the parameters `P`, `Q`, `pf`, `Vs`, and `V_dc` are defined as NumPy `ndarray` objects whose size determines the total simulation duration. Each element in these arrays represents a data point at a 1-second resolution, thereby defining the temporal resolution of the input profile used in the Python model.

4.1.6 Simulation Control and Model Management Parameters

This subsection defines parameters that control the execution, data management, and numerical time-stepping of the simulation. These variables are not directly related to the electro–thermal or electrical models but are essential for defining simulation structure, data storage behavior, and time discretization. They ensure consistent execution and reproducibility of the model under different operating scenarios.

Table 10: Simulation and model management parameters.

Variable	Type	Unit	Description
`saving_dataframes`	bool	–	Control flag to enable or disable saving of final simulation data as pandas DataFrames. *Note:* Inermediate results if required will still be saved to calculate final results.
`plotting_values`	bool	–	Boolean flag defining whether plots of electrical, thermal, and lifetime results are automatically generated at runtime.
`Location_dataframes`	str	–	Directory path where DataFrame files are stored when `saving_dataframes` is enabled.
`chunk_seconds`	int	s	Defines the simulation time window (in seconds) processed per computational chunk. This allows large mission profiles to be simulated efficiently in smaller time segments.
`dt`	float	s	Discrete simulation time step used for numerical integration. Determines the temporal resolution of the solver (e.g., 2 ms).
`Time_period`	property (int)	s	Total simulation duration expressed as the number of seconds.
`Nsec`	property (int)	–	Number of numerical time steps per second.
`Tgrid`	property (float)	s	Period of one full AC grid cycle.
`Ngrid`	property (int)	–	Number of simulation time steps per grid cycle.

4.2 mother_function.py

`mother_function.py` is the orchestration layer of the framework. It ingests the mission profile, executes power–flow and electro–thermal kernels, applies lifetime models, and emits analysis dataframes and figures. Two execution modes implement identical physics but different runtime strategies:

`main_1` (*single pass*) convenient for short missions (e.g., 24 h); minimal I/O, higher peak RAM.

`main_2` (*chunked*) streams long missions in time chunks to cap memory and accelerate post-processing; intermediate parquet files are merged at the end.

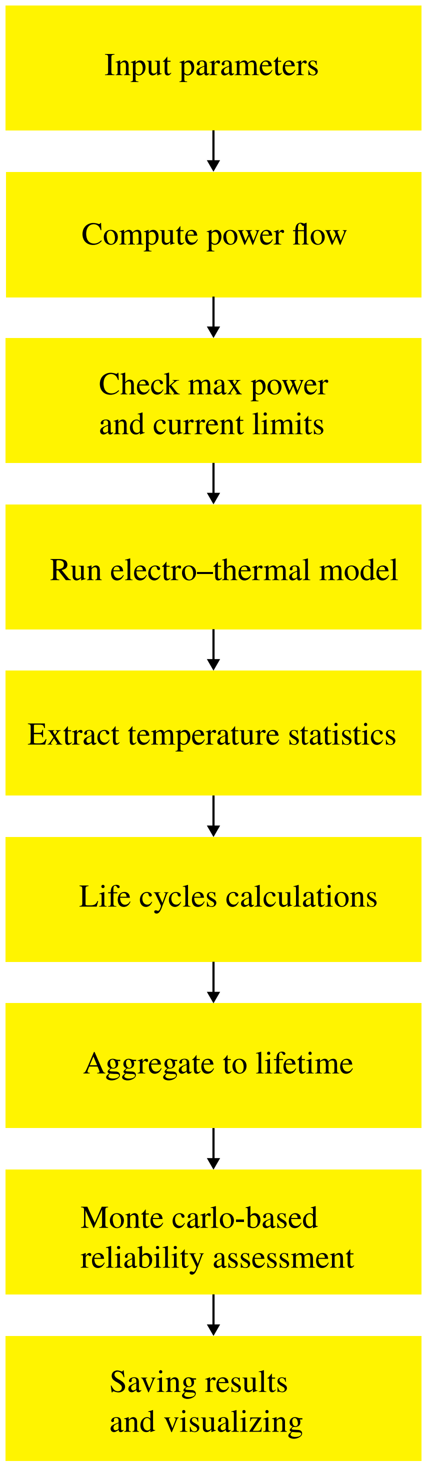
Input parameters
Compute power flow
Check max power and current limits
Run electro–thermal model
Extract temperature statistics
Life cycles calculations
Aggregate to lifetime
Monte carlo-based reliability assessment
Saving results and visualizing

Figure 2: Flowchart representation of basic physics of mother function file.

In Figure 2, the overall structure of the `mother_function.py` script is illustrated, showing how this file integrates the various submodels into a unified simulation framework. To provide an overview of the workflow, the user specifies input profiles of `P`, `Q`, and `pf`, each defined as a ndarray of length 86400, corresponding to a 24-hour (one-day) simulation. The simulation proceeds in discrete time steps, where each element of these arrays represents one second of real time.

Within each second, the inverter operates at a grid frequency of 50 Hz, corresponding to 50 electrical cycles per second. Consequently, one electrical period equals 0.02 s. For a defined simulation time step of `dt` = 0.001 s, each grid cycle is discretized into 20 points, enabling detailed resolution of the electrical waveforms over each fundamental period.

Because the electrical operating conditions (`P`, `Q`, `pf`) remain constant within a one-second interval, the electro–thermal behavior is computed for that interval as a steady-state representation. After completing the calculations for one second, the model proceeds to the next time step, repeating the process for the entire daily profile.

For computational efficiency, the simulation can be subdivided into larger processing segments defined by the parameter `chunk_seconds`. This variable specifies the duration (in seconds) of each data segment to be processed at once. For example, with `chunk_seconds = 86400`, the full 24-hour mission profile is simulated as a single block, whereas smaller values allow the model to execute and store results in multiple sequential chunks. This is important while carrying out 1 year long simulation as this approach significantly reduces memory usage during long-term or high-resolution simulations while maintaining continuity between successive time intervals.

The detailed implementation of the thermal model and its underlying Foster network formulation is described in Section 4.3.

4.2.1 Calling the mother function

The framework is executed through lightweight entry scripts that act as wrappers for the core routines `main_1()` and `main_2()` defined in `mother_function.py`. Each entry file corresponds to a specific simulation scenario—either *active-power* or *reactive-power* operation and selects between single-pass or chunked execution modes. The `main.py` script serves as the global launcher and invokes these entry files as separate Python processes via the helper function `run_once()`, ensuring that memory is fully released between runs. The user specifies the desired load profile by name, e.g.,

```
python main.py
```

which internally executes

```
python main_2_entry_active.py --profile alle_main_2_1_sec_inverter_1
```

as shown in Listing 1.

```python
# main.py
import subprocess
import sys

def run_once(entry_script, profile):
    """
    Run a given entry script as a completely new Python process.
    When the process exits, all its memory is freed.
    """
    print(f"\n Running {entry_script} for profile {profile}")
    cmd = [sys.executable, "-u", entry_script, "--profile", profile]
    subprocess.run(cmd, check=True)
    print(" Completed.\n")

if __name__ == "__main__":
    # Example call: 24-hour mission profile (main_2, active-power mode)
    run_once("main_2_entry_active.py", "alle_main_2_1_sec_inverter_1")
```

Listing 1: Example of calling the `mother_function.py` via the main launcher.

Each entry script reads the corresponding Parquet mission file located in `Load_profiles/`, extracts the arrays `P`, `Q`, and `pf`, and converts them to `numpy` vectors before passing them to the mother function. For instance, in `main_1_entry_reactive.py` and `main_2_entry_reactive.py`, the variables `P`, `Q`, and `pf` are loaded directly from the input file to reproduce the measured mission profile. Conversely, the `active` variants (`main_1_entry_active.py` and `main_2_entry_active.py`) overwrite these arrays with constant values to emulate a nominal active-power case at unity power factor. The argument `Loadprofile_name` is forwarded unchanged to ensure consistent labeling in saved results. The mother function then carries out the complete electro–thermal–lifetime computation chain and stores results in the `Finished_results/` and `Figures/` directories, as described in Section 4.5 and Section 4.6.

4.2.2 Additional intermediate and stochastic variables

In addition to the primary model inputs described earlier in Section 4.1, several auxiliary variables are defined within `mother_function.py`. These variables support the numerical execution of the electro–thermal lifetime model by maintaining state continuity, performing intermediate calculations or facilitating stochastic reliability analysis. Although not all of these variables are stored in the final dataframes, they are indispensable for the accurate operation of the model and for ensuring the physical consistency of results.

The electrical power–flow variables summarized in Table 11 represent the fundamental quantities governing the instantaneous operating state of the inverter within each one-second simulation interval. Together, they define the steady-state electrical boundary conditions at the inverter's AC terminals and form the primary inputs to the electro–thermal model. Specifically, `S` quantifies the total power magnitude transferred through the converter, `Is` establishes the corresponding RMS current stress on the semiconductor devices, and `phi` captures the relative phase displacement between voltage and current, thereby linking the electrical loading characteristics to the subsequent thermal and lifetime calculations.

Table 11: Electrical power–flow variables.

Variable	Type	Unit	Description
`S`	ndarray	VA	Per–sample apparent power on the AC side (one value per second sample). Computed consistently with `P`, `Q`, and `pf`.
`Is`	ndarray	A	Per–sample RMS current on the AC side. Interpreted as phase RMS for three–phase (PH = $V_{ll}/\sqrt{3}$) and line RMS for single–phase.
`phi`	ndarray	rad	Per–sample phase angle between AC voltage and current (`pf` $< 0 \Rightarrow$ inductive, current lags; `pf` $> 0 \Rightarrow$ capacitive, current leads; `pf` $= 0 \Rightarrow \pm\pi/2$).

The second group of variables as defined in Table 12 comprises temporary runtime parameters that exist only during the execution of the simulation or within the inner computational loops. These variables are used to track instantaneous thermal, electrical, and temporal states at each iteration and are essential for transient electro–thermal coupling. However, they are not written to the final dataframes since they serve only as intermediate containers required for calculating branch temperatures, instantaneous losses, and waveform histories during each time step.

Table 12: Temporary runtime variables used internally.

Variable	Type	Unit	Description
`Tbr_I`, `Tbr_D`	ndarray	K	Temperature rise of each Foster RC branch in the IGBT and diode junction–to–case thermal paths.
`Tbr_p`, `Tbr_s`	ndarray	K	Temperature rise across the interface paste and heat sink for shared thermal state.
`Tbr_p_I`, `Tbr_s_I`, `Tbr_p_D`, `Tbr_s_D`	ndarray	K	Corresponding temperature rise terms for separated thermal states of IGBT and diode.
`sec_idx_list`, `time_list`, `m_list`	list	–	Iteration index, time vector and modulation index lists.
`is_I_list`, `is_D_list`	list	A	IGBT and diode current waveforms.
`P_sw_I_list`, `P_sw_D_list`	list	W	Instantaneous switching power losses.
`P_con_I_list`, `P_con_D_list`	list	W	Instantaneous conduction power losses.
`P_leg_list`	list	W	Total instantaneous losses of one inverter leg.
`vs_list`, `is_inv_list`	list	V, A	Inverter AC-side voltage and current waveforms.
`TjI_list`, `TjD_list`	list	K	Time-series junction temperatures per second.
`TjI_all`, `TjD_all`	ndarray	K	Concatenated temperature arrays across the full mission time.

The third category contains deterministic intermediate variables that bridge the electro–thermal simulation and lifetime estimation processes as defined in Table 13. These quantities are obtained after completing the transient temperature calculations and before performing Monte Carlo randomization. They describe the mean junction temperatures, temperature swings, heating durations, and corresponding cycles-to-failure that form the physical basis for calculating component degradation and lifetime consumption.

Table 13: Intermediate variables used in temperature cycle extraction and lifetime estimation.

Variable	**Type**	**Unit**	**Description**
`TjI_mean`, `TjD_mean`	ndarray	K	Mean junction temperatures computed over defined time windows for IGBT and diode.
`TjI_delta`, `TjD_delta`	ndarray	K	Temperature swings (ΔT_j) for each heating cycle.
`t_cycle_heat_I`, `t_cycle_heat_D`	ndarray	s	Heating durations of each temperature cycle.
`Nf_I`, `Nf_D`	ndarray	–	Cycles to failure computed from LESIT empirical model.
`Life_I`, `Life_D`	float	years	Estimated lifetimes of IGBT and diode based on cumulative thermal fatigue.
`number_of _yearly_cycles`	float	–	Total number of thermal cycles per year derived from mission profile.
`Yearly_life _consumption_I`, `Yearly_life _consumption_D`	float	–	Fraction of component lifetime consumed per operational year.
`Tj_mean_float_I`, `Tj_mean_float_D`	float	K	Average junction temperatures used in lifetime model simplification.
`delta_Tj_float_I`, `delta_Tj_float_D`	float	K	Corresponding mean temperature swings.
`t_cycle_float`	float	s	Representative heating time constant used in later Monte Carlo analysis.

The last category of variables, as defined in Table 14, corresponds to the Monte Carlo-based reliability assessment stage. This stochastic analysis is performed to quantify the uncertainty inherent in the electro–thermal lifetime prediction process. Since parameters such as thermal resistance, activation energy, and empirical constants in the LESIT model are derived from experimental characterization, they naturally exhibit statistical variation across manufacturing batches and operating environments. The Monte Carlo approach propagates these uncertainties by repeatedly sampling the lifetime model inputs from normal distributions and recalculating the expected lifetime for each realization. This method provides a probabilistic representation of device reliability rather than a single deterministic value, allowing the estimation of confidence intervals, mean time to failure, and lifetime dispersion for both the IGBT and diode. Such statistical evaluation is crucial for assessing inverter dependability under realistic operating variability, as also emphasized by Scheuermann [3] and by recent studies on PV inverter lifetime prediction [4].

Table 14: Variables used for Monte Carlo–based lifetime and reliability assessment.

Variable	Type	Unit	Description
`A_normal_distribution`, `alpha_normal_distribution`, `beta0_normal_distribution`, `beta1_normal_distribution`, `C_normal_distribution`, `gamma_normal_distribution`, `fd_normal_distribution`, `fI_normal_distribution`, `Ea_normal_distribution`, `ar_normal_distribution`, `k_b_normal_distribution`	ndarray	–	Normally distributed samples of the LESIT lifetime model parameters used for uncertainty propagation.
`t_cycle_float_normal_distribution`, `Tj_mean_float_I_normal_distribution`, `delta_Tj_float_I_normal_distribution`, `Tj_mean_float_D_normal_distribution`, `delta_Tj_float_D_normal_distribution`	ndarray	–	Randomized distributions of thermal cycle parameters for IGBT and diode used in probabilistic failure estimation.
`Nf_I_normal_distribution`, `Nf_D_normal_distribution`	ndarray	–	Stochastic estimates of cycles-to-failure for IGBT and diode.
`Life_period_I_normal_distribution`, `Life_period_D_normal_distribution`, `Life_period_switch_normal_distribution`	ndarray	years	Resulting lifetime distributions for IGBT, diode, and overall switch pair.

4.3 Electro_thermal_behavior_file.py

The `Electro_thermal_behavior_file.py` module constitutes the core of the electro–thermal coupling framework. Its primary objective is to translate instantaneous electrical power losses of the inverter semiconductors IGBT-Diode pair into transient junction temperature trajectories, using a discretized Foster thermal network representation. This temperature evolution forms the physical foundation for subsequent lifetime estimation and reliability analysis.

4.3.1 Variables introduced

In addition to the electrical and thermal parameters defined in the Section 4.1 and Section 4.2, the `Electro_thermal_behavior_file.py` introduces a set of new runtime variables that describe the instantaneous electro–thermal state of the inverter leg. These variables are generated and updated within each one–second simulation window as part of the coupling process between electrical losses and thermal response. The newly introduced variables are summarized in Table 15, where their data type, physical unit, and purpose are detailed.

Table 15: New Variables used in file `Electro_thermal_behavior_file.py`.

Variable	**Type**	**Unit**	**Description**
`t`	ndarray	s	Time vector over the simulated 1 s window.
`m`	ndarray	-	Instantaneous modulation.
`is_I`	ndarray	A	Instantaneous IGBT current.
`is_D`	ndarray	A	Instantaneous diode current.
`P_sw_I`	ndarray	W	IGBT switching loss power.
`P_sw_D`	ndarray	W	Diode switching loss power.
`P_con_I`	ndarray	W	IGBT conduction loss power.
`P_con_D`	ndarray	W	Diode conduction loss power.
`P_I`	ndarray	W	Total IGBT losses that is `P_sw_I` + `P_con_I`.
`P_D`	ndarray	W	Total Diode losses that is `P_sw_D` + `P_con_D`.
`P_leg`	ndarray	W	Total inverter leg loss that is `P_I` + `P_D` .
`T_j_I`	ndarray	K	IGBT junction temperature trajectory.
`T_j_D`	ndarray	K	Diode junction temperature trajectory.
`vs_inverter`	ndarray	V	Instantaneous AC-side phase voltage.
`is_inverter`	ndarray	A	Instantaneous AC-side phase current.

4.3.2 Thermal Network Formulation

Each semiconductor and its corresponding cooling path are modeled by a Foster-type thermal RC network, consisting of N parallel branches with time constants τ_i and resistances r_i. The thermal state of each branch evolves according to an exponentially decaying kernel. For a given branch i, the discrete-time temperature contribution is expressed as:

$$T_{\mathrm{br},i}(t+\Delta t) = \alpha_i\, T_{\mathrm{br},i}(t) + (1-\alpha_i)\, r_i\, P(t), \tag{5}$$

where $\alpha_i = e^{-\Delta t/\tau_i}$ is the discrete decay factor, r_i is the thermal resistance in K/W, and $P(t)$ denotes the instantaneous device power loss (conduction and switching components). The total junction temperature is obtained as the sum of all branch contributions and the ambient temperature T_{amb}:

$$T_j(t) = T_{\mathrm{amb}} + \sum_{i=1}^{N} T_{\mathrm{br},i}(t). \tag{6}$$

This approach allows transient heat accumulation and dissipation to be represented efficiently in a numerically stable form. The Foster network parameters (r_i, τ_i) are typically identified from manufacturer transient thermal impedance curves.

4.3.3 Shared and Separated Thermal States

Two distinct topologies are implemented:

1. **Shared Thermal State:** The IGBT and diode share the same thermal path for the paste and heatsink layers, representing a common cooling interface. This configuration is modeled by the function `thermal_rollout_shared_thermal_state()`.

2. **Separated Thermal State:** Each device possesses an independent paste and heatsink thermal network. This allows asymmetric temperature distributions between the IGBT and the diode, as implemented in `thermal_rollout_separated_thermal_state()`.

Both implementations propagate the thermal RC states over a one-second simulation window, corresponding to the smallest electrical kernel duration in the inverter mission profile.

4.3.4 Numerical Implementation

The thermal rollout routines are compiled using the `Numba` `@njit` decorator to achieve near-native computational performance. This is critical for long-term mission profile simulations involving several thousand thermal cycles. Each rollout function operates on pre-allocated `NumPy` arrays, ensuring memory contiguity and cache efficiency. Prior to thermal propagation, electrical quantities are precomputed and cached through the helper functions `_kernel_key_tuple()` and `_build_kernel_one_second_cached()`. These utilize the `functools.lru_cache` mechanism to store previously generated electrical waveforms, thereby eliminating redundant recalculations for identical operating conditions.

4.3.5 Electro-Thermal Simulation Flow

The high-level class `Electro_thermal_behavior_class` encapsulates the coupling procedure through two main methods:

- `Electro_thermal_behavior_shared_thermal_state()`
- `Electro_thermal_behavior_separated_thermal_state()`

Each method executes the following sequence for every one-second simulation window:

1. **Electrical kernel generation:** Using inverter operating parameters $(V_s, I_s, \phi, V_{dc}, M, f_{sw})$, the instantaneous voltage, current, and power loss waveforms are produced through calls to the `Calculation_functions_class`.
2. **Thermal state advancement:** The appropriate rollout function (shared or separated) is invoked to propagate the thermal RC network according to Equations (5)–(6).
3. **Output generation:** The resulting IGBT and diode junction temperatures $T_{j,\mathrm{IGBT}}(t)$ and $T_{j,\mathrm{D}}(t)$, together with all intermediate quantities, are returned to the higher-level control routine (`mother_function.py`).

The branch temperature states T_{br} are preserved between successive calls, enabling long-term transient thermal simulations with cumulative heating and cooling effects.

4.3.6 Physical Interpretation

The electro-thermal behavior module thus bridges the electrical and thermal domains through a bidirectional coupling. Electrical operating conditions determine instantaneous heat generation, while the thermal network defines how this energy propagates through the semiconductor stack. This interaction governs the time-dependent junction temperature:

$$T_j(t) = T_{\text{amb}} + \Delta T_{j\to c}(t) + \Delta T_{c\to s}(t) + \Delta T_{s\to a}(t), \tag{7}$$

where $\Delta T_{j\to c}$, $\Delta T_{c\to s}$, and $\Delta T_{s\to a}$ represent temperature rises across the junction–case, case–sink, and sink–ambient interfaces, respectively. These temperature histories are subsequently employed to compute the thermal stress amplitude ΔT_j and mean junction temperature $\overline{T}_j$ required for the lifetime evaluation model.

4.3.7 Computational Advantages

The combination of JIT compilation and kernel caching allows this module to process thousands of one-second simulation windows efficiently, making it suitable for year-long mission profile analyses. The modular structure further supports flexible thermal configurations and parameter substitutions for different semiconductor packages or cooling designs.

In summary, the `Electro_thermal_behavior_file.py` module forms the computational backbone of the electro-thermal lifetime assessment framework. It delivers physically accurate, numerically efficient junction temperature estimations under arbitrary inverter operating conditions, thus enabling robust and scalable reliability prediction.

4.4 Calculation_functions_file.py

The `Calculation_functions_file.py` module serves as the computational core of the electro–thermal and reliability modeling framework. It provides a structured collection of static methods that implement the analytical and numerical equations governing inverter operation, semiconductor device losses, thermal stress cycling, and lifetime estimation. The class `Calculation_functions_class` encapsulates all physics-based routines—ranging from instantaneous voltage and current generation to switching and conduction loss computation, thermal window analysis, and reliability prediction using Miner's rule and Monte Carlo uncertainty propagation. Each function is designed to operate independently, ensuring numerical efficiency and clear traceability within the larger simulation architecture. Below most of the important functions are explained along with the physics behind those function.

compute_power_flow_from_pf

Depending on the inverter configuration, the function first determines the theoretical maximum RMS output voltage (`Vs_theoretical`) that the inverter can produce based on the DC-link voltage (`V_dc`), modulation index (`M`), number of phases (`inverter_phases`), and modulation strategy (`modulation_scheme`). For single-phase inverters, the value also depends on whether a *half-bridge* or *full-bridge* topology (`single_phase_inverter_topology`) is used. The corresponding expressions are:

$$Vs\_theoretical = \begin{cases} \dfrac{M\,V\_dc}{\sqrt{2}}, & \text{single-phase full-bridge} \\ \dfrac{M\,V\_dc}{2\sqrt{2}}, & \text{single-phase half-bridge} \\ \dfrac{M\,V\_dc}{\sqrt{6}}, & \text{three-phase svm (Space Vector PWM)} \\ \dfrac{M\,V\_dc}{2\sqrt{2}}, & \text{three-phase spwm (Sinusoidal PWM)} \end{cases} \tag{8}$$

If the user does not provide an AC-side RMS voltage (`Vs`), it is automatically set equal to `Vs_theoretical`. The function also performs a consistency check to ensure that `Vs` does not exceed its theoretical limit, as this would imply an invalid modulation or overmodulation condition.

Furthermore, the function calculates the apparent power (`S`), RMS current (`Is`), and phase angle (`phi`) for each simulation step based on the active power (`P`), reactive power (`Q`), power factor (`pf`), and inverter operating conditions. It ensures that the AC-side voltage (`Vs`) remains within its theoretical modulation-dependent limit and updates `P` and `Q` accordingly. For a given time step, the apparent power, RMS current,

and phase angle are computed as:

$$S = \frac{P}{|pf|}, \quad I\_s = \frac{S}{V\_s}, \quad \phi = \begin{cases} -\arccos(|pf|), & pf < 0(inductive) \\ \arccos(|pf|), & pf > 0(capacitive) \\ \pm\frac{\pi}{2}, & pf = 0 \end{cases} \tag{9}$$

Finally, the reactive power is updated to maintain consistency with the apparent and active power values:

$$Q = \pm\sqrt{S^2 - P^2} \tag{10}$$

compute_power_flow_from_pf_design_control_inverter

This function performs the same fundamental calculations as `compute_power_flow_from_pf`, determining the apparent power (`S`), RMS current (`Is`), and phase angle (`phi`) based on `P`, `Q`, and `pf`. The key distinction is the introduction of the variable `N_parallel`, which can automatically determines the number of power semiconductor devices connected in parallel required to satisfy current and power limits. It can also take user input for `N_parallel`.
The number of parallel switches is computed from the ratio between the system apparent power and the effective maximum device capacity, including a safety margin (`overshoot_margin_inverter`):

$$N\_parallel = \left\lceil \frac{\max(S)}{inverter\_phases \times Vs \times max\_IGBT\_RMS\_Current} \right. \\ \left. \times \frac{1}{(1 + overshoot\_margin\_inverter)} \right\rceil \tag{11}$$

This ensures that each IGBT diode switch pair operates within its rated RMS current limit. Once `N_parallel` is determined, the per-device power values (`P`, `Q`, and `S`) are scaled accordingly before performing the same power flow and phase calculations as in function `compute_power_flow_from_pf`.

Inverter_voltage_and_current

This function computes the instantaneous inverter phase voltage (`vs_inverter`) and current (`is_inverter`) waveforms from their RMS values and the phase angle between them [2]. Using the grid angular frequency (`omega`) and time vector (`t`), it reconstructs the sinusoidal waveforms that represent the inverter's AC-side operation. The voltage and current are calculated as:

$$vs\_inverter = \sqrt{2}\, Vs \, \sin(omega \cdot t) \tag{12}$$

$$is\_inverter = \sqrt{2}\,Is\;\sin(omega \cdot t + phi) \tag{13}$$

Here, `phi` represents the phase displacement between voltage and current, where a negative value indicates an inductive load (current lags voltage) and a positive value indicates a capacitive load (current leads voltage). The resulting waveforms are returned as NumPy arrays corresponding to the instantaneous AC-side signals of the inverter.

Instantaneous_modulation

This function computes the instantaneous modulation signal (`m`) used to describe the inverter's switching behavior over time [2]. The modulation index (`M`) defines the amplitude of the sinusoidal reference waveform relative to the DC-link voltage, while the term (`omega` · `t` + `phi`) represents its instantaneous phase.
The instantaneous modulation function is expressed as:

$$m = \frac{M\;\sin(omega \cdot t + phi) + 1}{2} \tag{14}$$

This formulation normalizes the modulation waveform between 0 and 1, which is suitable for pulse-width modulation (PWM) schemes. The modulation signal thus defines the instantaneous duty ratio of the inverter's switching devices, directly influencing the output voltage waveform and harmonic content.

IGBT_and_diode_current

This function computes the instantaneous current through the IGBT (`is_I`) and its antiparallel diode (`is_D`) within a single inverter leg [2]. The calculation uses the inverter's RMS output current (`Is`), instantaneous modulation signal (`m`), and grid angular frequency (`omega`) to determine the current waveform over time.
The instantaneous base current is first defined as:

$$base = \sqrt{2}\,Is\;\sin(omega \cdot t)\,m \tag{15}$$

Since power electronic switches conduct unidirectionally, the IGBT current corresponds to the positive portion of the waveform, while the diode current corresponds to the negative portion. These are extracted using:

$$is\_I = \max(base, 0), \quad is\_D = \max(-base, 0) \tag{16}$$

This ensures that the IGBT carries current only during its conduction interval (positive half-cycles), while the diode conducts during the complementary negative intervals, accurately reflecting the switching leg's operation in a half-bridge configuration.

Switching_losses

This function computes instantaneous switching power losses for the IGBT (`P_sw_I`) and the antiparallel diode (`P_sw_D`) [2]. The IGBT loss is modeled from turn-on/turn-off overlap energy, proportional to `V_dc`, device current, and switching transition times `t_on`, `t_off`. The diode loss is based on reverse-recovery energy `Err_D` scaled to the operating `V_dc` and current. Energies per event are multiplied by the switching frequency `f_sw` to obtain power, and negative values are clipped to zero.

$$P\_sw\_I = \left[\frac{\sqrt{2}}{2\pi} V\_dc\, is\_I(t)\, t\_on + \frac{\sqrt{2}}{2\pi} V\_dc\, is\_I(t)\, t\_off \right] f\_sw \tag{17}$$

$$P\_sw\_D = \left[\frac{\sqrt{2}}{\pi} \frac{is\_D(t)\, V\_dc}{I\_ref\, V\_ref} \right] Err\_D\, f\_sw \tag{18}$$

Conduction_losses

This function calculates the instantaneous conduction losses of the inverter's IGBT (`P_con_I`) and diode (`P_con_D`) based on their effective on-resistance and knee voltage [2]. The model accounts for both resistive and threshold voltage contributions, as well as modulation index (`M`) and power factor (`pf`) effects, which influence the average conduction interval of each device. Negative values are clipped to zero to ensure physical validity.

$$P\_con\_I = \frac{is\_I(t)^2}{4} R\_IGBT + \frac{is\_I(t)}{\sqrt{2\pi}} V\_0\_IGBT + \left[\frac{is\_I(t)^2}{4} \frac{8\,M}{3\pi} R\_IGBT + \frac{is\_I(t)}{\sqrt{2\pi}} \frac{\pi\,M}{4} V\_0\_IGBT \right] |pf| \tag{19}$$

$$P\_con\_D = \frac{is\_D(t)^2}{4} R\_D + \frac{is\_D(t)}{\sqrt{2\pi}} V\_0\_D - \left[\frac{is\_D(t)^2}{4} \frac{8\,M}{3\pi} R\_D + \frac{is\_D(t)}{\sqrt{2\pi}} \frac{\pi\,M}{4} V\_0\_D \right] |pf| \tag{20}$$

Cycles_to_failure

This function evaluates the cycles-to-failure `Nf` per thermal cycle using Equation 1 as described in [2] and [3] The model reads:

$$Nf = A\ (delta\_Tj)^{alpha}\ (ar)^{beta1 \cdot delta\_Tj + beta0}\ \frac{C + (t\_cycle\_heat)^{gamma}}{C + 1}\ \exp\!\left(\frac{Ea}{k\_b\, Tj\_mean} \right) fd. \tag{21}$$

A basic validity check enforces `Tj_mean` > 0 K.

window_stats

This function computes per-window thermal statistics from a junction-temperature series `temp`. With a window length `time_window` (s) and resolution `steps_per_sec`, the window size is

$$window\_size = \mathrm{round}(time\_window \times steps\_per\_sec), \tag{22}$$

and the signal is reshaped into contiguous windows. For each window, it returns: the mean temperature,

$$mean\_T = \text{mean}\big(temp_{\text{window}}\big), \tag{23}$$

the swing as max–min,

$$delta\_T = \max\big(temp_{\text{window}}\big) - \min\big(temp_{\text{window}}\big), \tag{24}$$

and the heating duration estimated from the index distance between the maximum and minimum within the window,

$$t\_cycle\_heat = \frac{\big|\arg\max(temp_{\text{window}}) - \arg\min(temp_{\text{window}})\big|}{steps\_per\_sec}. \tag{25}$$

The array `time_period_df2` provides the window start indices aligned to `pf`.
Note. The outputs (`mean_T`, `delta_T`, `t_cycle_heat`) are window-based extrema statistics that act as inputs to the lifetime model. This has been done to reduce computation time as a full rainflow counting procedure [4] is computational intensive. The method mentioned here has some limitations but it works fine in our case.

check_vce

This function verifies that the collector–emitter voltage of the IGBT does not exceed its rated maximum limit. The instantaneous collector–emitter voltage (`V_CE`) is computed from the DC-link voltage (`V_dc`) and a user-defined overshoot margin (`overshoot_margin`), which accounts for transient voltage spikes during switching:

$$V\_CE = V\_dc \times (1 + overshoot\_margin) \tag{26}$$

If any calculated value of `V_CE` exceeds the maximum allowable limit (`max_V_CE`), the function raises a `ValueError` to indicate that the operating condition violates the device voltage rating. This safety check ensures that the inverter operates within the semiconductor's safe operating area, preventing overvoltage-induced breakdown or permanent damage.

check_igbt_diode_limits

This function verifies that the instantaneous and RMS values of current and junction temperature for both the IGBT and diode remain within their specified operating limits. It acts as a validation layer in the electro–thermal model, ensuring that the simulated conditions remain within the safe operating area of the devices. The RMS and peak currents for the IGBT and diode are calculated using the following relations:

$$I\_rms\_is\_I = \sqrt{\text{mean}(is\_I^2)} \tag{27}$$

$$I\_peak\_is\_I = \max(|is\_I|) \tag{28}$$

$$I\_rms\_is\_D = \sqrt{\text{mean}(is\_D^2)} \tag{29}$$

$$I\_peak\_is\_D = \max(|is\_D|) \tag{30}$$

Each computed value is then compared against the corresponding rated limits that is `max_IGBT_RMS_Current`, `max_IGBT_peak_Current`, `max_Diode_RMS_Current`, `max_Diode_peak_Current`, `max_IGBT_temperature`, and `max_Diode_temperature`. If any threshold is exceeded, the function raises a `ValueError`, prompting corrective measures such as load reduction, enhanced cooling, or higher-rated device selection. By enforcing these constraints, the model maintains realistic thermal and electrical behavior throughout the simulation, ensuring numerical stability and physical consistency in the lifetime prediction framework.

delta_t_calculations

This function estimates an equivalent steady-state temperature swing (`delta_Tj_float`) for monte carlo reliability assessment as explained in Section 4.2. It uses the lifetime model formulation as explained in 1 inverted through the Lambert–*W* function to obtain a representative temperature swing from cycle-based lifetime data. Additionally, it computes auxiliary quantities such as the number of yearly thermal cycles and annual life consumption fraction.

The total number of equivalent thermal cycles per year and the yearly life consumption fraction are given by:

$$number\_of\_yearly\_cycles = 3600 \times 24 \times 365 \times \left(\frac{\text{len}(Nf)}{\text{len}(pf)}\right) \times f \tag{31}$$

$$Yearly\_life\_consumption\_I = \frac{1}{Life} \tag{32}$$

The mean junction temperature used:

$$Tj\_mean\_float = \text{mean}(Tj\_mean) \tag{33}$$

An intermediate constant is then defined as:

$$K = A\left(\frac{C + (t\_cycle\_float)^{gamma}}{C+1}\right)\exp\left(\frac{Ea}{k\_b\,Tj\_mean\_float}\right) f d\, ar^{beta0} \tag{34}$$

and the normalized lifetime parameter:

$$N_* = \frac{number\_of\_yearly\_cycles}{Yearly\_life\_consumption\_I} \tag{35}$$

Finally, the equivalent temperature swing (principal branch W_0) is computed as:

$$delta\_Tj\_float = \frac{alpha}{beta1\ \ln(ar)} W_0\left(\frac{beta1\ \ln(ar)}{alpha}\left(\frac{N_*}{K}\right)^{1/alpha}\right) \tag{36}$$

The function returns `number_of_yearly_cycles`, `Yearly_life_consumption_I`, `Tj_mean_float`, `delta_Tj_float`, and `t_cycle_float`.

check_max_apparent_power_switch

This function ensures that the inverter's apparent power demand (`S`) does not exceed the maximum capability of the semiconductor devices based on their RMS current rating. For each simulation step, the maximum allowable apparent power is computed as:

$$S\_max = Vs \times max\_IGBT\_RMS\_Current \times inverter\_phases \tag{37}$$

If any instantaneous value of `S` exceeds this limit, the function raises a `ValueError`, prompting the user to reduce the inverter loading or choose devices with higher current ratings. This check ensures that the inverter operates within its safe current-carrying capability, preventing device overstress and maintaining reliability.

Lifecycle_calculation_acceleration_factor

This function estimates the mission-referenced lifetime of a semiconductor device based on Miner's cumulative damage rule and an acceleration factor approach. It converts per-cycle fatigue data (`Nf`) into an equivalent calendar lifetime (`Life`) under a given power factor profile (`pf`). The computation incorporates both load-induced and calendar-based degradation mechanisms.

First, the cumulative per-set damage is computed as:

$$damage\_per\_set = \sum_i \frac{1}{N_{f,i}} \tag{38}$$

where each $N_{f,i}$ represents the number of cycles to failure for the i-th thermal event within one operating window (i.e., one second).

Next, the time normalization converts the damage per operating window into a yearly reference using:

$$sets\_per\_year = \frac{3600 \times 24 \times 365}{\text{len}(pf)} \tag{39}$$

and introduces a calendar-based minimum damage floor to prevent infinite lifetime estimation under low-stress operation:

$$floor\_damage\_per\_set = \frac{1}{sets\_per\_year \times Component\_max\_lifetime} \tag{40}$$

The total effective per-set damage is therefore:

$$effective\_damage\_per\_set = damage\_per\_set + floor\_damage\_per\_set \tag{41}$$

The acceleration factor compares the effective per-set damage to the nominal mission damage rate:

$$AF = \frac{effective\_damage\_per\_set}{target\_damage\_per\_set} \quad where \quad target\_damage\_per\_set = \frac{1/Component\_max\_lifetime}{sets\_per\_year} \tag{42}$$

Finally, the equivalent lifetime (in years) is obtained as:

$$Life = \frac{Component\_max\_lifetime}{AF} \tag{43}$$

This formulation ensures that the estimated lifetime converges to the rated mission life under nominal loading and reduces proportionally for harsher thermal or electrical stress conditions. The added calendar floor term avoids nonphysical infinite lifetimes at extremely mild operating conditions.

Lifecycle_normal_distribution_calculation_acceleration_factor

This function follows the same formulation as `Lifecycle_calculation_acceleration_factor`, based on Miner's cumulative damage rule and acceleration factor methodology. However, it is specifically used within the Monte Carlo–based reliability assessment, where the lifetime model parameters are randomly sampled from normal distributions to account for manufacturing and material variability.

Here the end result `Life_period` is similar to function `Lifecycle_calculation_acceleration_factor` but here `Life_period` represents the probabilistic lifetime outcome for each Monte Carlo sample, forming the basis for the overall statistical lifetime distribution of the IGBT and diode.

Apart from the physics-based and reliability-related computations, the module also includes several auxiliary functions designed to improve numerical efficiency, memory management, and data handling. These functions do not directly contribute to the physical modeling but are essential for enabling large-scale, long-duration simulations to run efficiently. The functions include: `downsample_arrays_df_2`, `infer_sample_rate`, `prepare_df2_and_time_axis`, `write_dataframe_fast`, `resize_foster_branches`, `load_latest_df`, `latest_chunk_file`, `merge_parquet_files`, `find_sorted_files`, `cat`, and `free_ram_now`. Together, these routines manage tasks such as dynamic memory release, efficient I/O operations with `pandas` dataframes, adaptive time-resolution control, and the merging of simulation outputs across multiple time chunks—allowing the full electro–thermal simulation to remain computationally stable and scalable.

4.5 Dataframe_saving_file.py

The `Dataframe_saving_file.py` module handles the structured storage of simulation outputs generated by the electro–thermal lifetime model. It defines a single core function, `save_dataframes()`, which receives four dataframes (`df_1`–`df_4`) representing distinct categories of computed results such as instantaneous electrical losses, thermal responses, aggregated operating conditions, and lifetime indicators. For each simulation, a unique directory is created based on a timestamp identifier, ensuring chronological organization and reproducibility of stored data. The module employs the `pandas` and `pyarrow` libraries to serialize data into the highly efficient `Parquet` format with Zstandard (`zstd`) compression (level 7) and dictionary encoding, balancing compact storage with rapid read/write performance. In addition, the module utilizes the `write_dataframe_fast()` method from the `Calculation_functions_class` to accelerate file output for large datasets. This design guarantees consistent, high-throughput storage of simulation results, facilitating subsequent visualization, statistical evaluation, and reliability analysis.

4.6 Plotting_file.py

To facilitate interpretation of the electro–thermal and lifetime analysis, the developed Python framework automatically generates a comprehensive set of plots that summarize the intermediate and final results of the simulation. Each figure corresponds to a specific physical or reliability quantity derived from the coupled electro–thermal model, as detailed below.

4.6.1 Temperature and Thermal Cycling Behaviour (Figs. 1–3, 17)

- **Figure 1 – IGBT and Diode Mean Junction Temperature:** Displays the evolution of the mean junction temperature $T_{j,\text{mean}}$ for both the IGBT and diode across the mission profile. It provides direct insight into the steady-state thermal operating level of each device and the impact of loading and ambient temperature.

- **Figure 2 – IGBT and Diode Temperature Swing:** Shows the instantaneous temperature variation ΔT_j for each device. The amplitude of these swings is proportional to thermal fatigue severity and forms a key input to the lifetime model.

- **Figure 3 – Cycles to Failure:** Depicts the calculated number of thermal cycles to failure N_f for the IGBT and diode using the empirical LESIT lifetime law. The logarithmic scale highlights the exponential dependence of fatigue life on temperature swing and mean temperature.

- **Figure 17 – Heating Duration of Each Thermal Cycle:** Illustrates the computed heating time $t_{\text{cycle,heat}}$ for the IGBT and diode, representing the duration of high-stress operation within each cycle.

4.6.2 Lifetime Evaluation (Figs. 18–26)

- **Figure 18 – Individual Device Lifetime:** Bar chart comparing the total expected lifetime (in years) of the IGBT and diode under the given mission profile.

- **Figure 19 – Combined Switch Lifetime:** Shows the equivalent lifetime of the IGBT–diode pair (switch leg), derived as the weaker of the two components.

- **Figures 20–22 – Lifetime Distributions:** Histograms of the probabilistic lifetime distribution obtained from Monte-Carlo sampling for the IGBT, diode, and switch. These reflect manufacturing variability and uncertainty in material parameters.

- **Figures 23–25 – Cumulative Distribution Functions (CDFs):** Corresponding cumulative probability plots for the same distributions, providing the probability of failure as a function of operating time.

- **Figure 26 – Inverter System Lifetime CDF:** Presents the system-level lifetime distribution for the full inverter, accounting for the number of series positions (four for single-phase, six for three-phase) and parallel devices per switch leg. System failure is defined by the earliest device failure across all components.

4.6.3 Electrical and Instantaneous Waveform Analysis (Figs. 4–10)

- **Figure 4 – Instantaneous Modulation Signal:** Displays the normalized modulation waveform $m(t)$ for one fundamental electrical cycle, illustrating PWM duty variation.
- **Figure 5 – Instantaneous IGBT and Diode Currents:** Shows the conduction intervals of each device within one switching leg, confirming correct current sharing between the transistor and its antiparallel diode.
- **Figures 6–7 – Switching and Conduction Losses:** Present instantaneous loss waveforms $P_{\text{sw}}(t)$ and $P_{\text{con}}(t)$ for the IGBT and diode. These verify energy dissipation patterns and allow evaluation of dominant loss mechanisms.
- **Figure 8 – Inverter Phase Voltage and Current:** Illustrates the AC-side electrical waveforms over one cycle, providing a visual check of phase displacement corresponding to the power factor.
- **Figure 9 – Instantaneous Junction Temperature (One Cycle):** Shows transient junction temperature oscillations of both devices during one representative cycle, highlighting short-term electro–thermal coupling.
- **Figure 10 – Total Power Loss per Switching Leg:** Summarizes the combined instantaneous losses of the IGBT–diode pair, representing total inverter leg dissipation.

4.6.4 Electrical Operating Conditions (Figs. 12–16, 27–28)

- **Figure 12 – Active, Reactive, and Apparent Power:** Time-series plot of P, Q, and S across the mission profile, showing how the inverter shares active and reactive power under different load conditions.
- **Figure 13 – Power Factor:** Depicts the instantaneous power factor, indicating the reactive power support or absorption level.
- **Figure 14 – Phase Angle:** Shows the corresponding current–voltage phase displacement in degrees, confirming the relation between power factor and phase shift.
- **Figure 15 – RMS Voltage and Current:** Plots inverter RMS terminal voltage and current, providing a steady-state electrical summary of each operating point.
- **Figure 16 – DC-Link Voltage:** Displays the DC-bus voltage supplied to the inverter throughout the mission profile.
- **Figure 27 – Number of Switches in Parallel:** Bar chart indicating how many semiconductor devices operate in parallel per leg, as determined by the design control algorithm.
- **Figure 28 – Execution Time:** Reports the total computational time required for a complete simulation, expressed in seconds, minutes, or hours depending on run duration.

Collectively, these plots provide a complete visualization chain—from instantaneous electrical behaviour to cumulative reliability statistics. Early figures (Figs. 4–10) validate the physical accuracy of the inverter and semiconductor models, mid-range figures (Figs. 1–3, 12–17) describe electro–thermal loading and stress evolution, and the final figures (Figs. 18–26) quantify probabilistic lifetime and system reliability. This structured visualization enables both qualitative verification of model correctness and quantitative assessment of device endurance under realistic mission profiles.

5 Model execution and usage guide

This section provides a concise user-oriented guide for executing the electro–thermal lifetime assessment framework. While Section 4.2.1 describes the internal structure and calling sequence of the code, the present section focuses on practical instructions for setting up and running the model.

5.1 Python Environment Setup

The framework has been developed and tested under **Python 3.13.7**. All required dependencies are listed in the file `requirements.txt` located in the main repository directory. To install these dependencies, create a virtual environment and execute:

```
pip install -r requirements.txt
```

The model is compatible with Windows 10/11, Linux (Ubuntu 20.04 or newer), and macOS operating systems.

5.2 Running the Model

1. **Configure Input Parameters**
   Open the file `Input_parameters_file.py` and define the following:
   - Inverter and semiconductor parameters (rated power, voltages, current limits, etc.).
   - Thermal RC network coefficients for IGBT and diode.
   - Lifetime model constants, mission-profile settings, and simulation control flags such as `saving_dataframes` and `plotting_values`.
2. **Load or Create a Mission Profile**
   Prepare a mission profile file and place it inside the directory `/Load_profiles/`. Mission profiles may be generated using `Load_profile_builder.py` or imported as existing `.parquet` or `.csv` files.
3. **Execute the Simulation**
   Run the model from a terminal or command prompt using one of the following commands:

   ```
   python main.py
   ```

   or specify a direct entry script for a specific case:

   ```
   python main_2_entry_active.py --profile example_profile
   ```

   The script automatically calls the corresponding `mother_function.py` and executes the complete electro–thermal–lifetime workflow.
4. **View Results**
   Upon successful execution, all numerical outputs are stored in:

- `/Finished_results/` — contains `.parquet` files of the processed DataFrames, including temperature profiles, power losses, and lifetime statistics.
- `/Figures/` — contains automatically generated plots of junction temperature evolution, power loss characteristics, and lifetime consumption trajectories.

5. **Interpretation of Key Outputs**
   The primary simulation results are focused on the lifetime prediction and reliability evaluation of the inverter semiconductor devices. These results quantify how the combined electro–thermal stresses affect the expected operating life of both the IGBT and diode. The most relevant outputs are summarized below and correspond to Figures 18–26.

5.3 Typical Runtime and Computational Considerations

The computational performance and memory footprint of the developed electro–thermal lifetime assessment model strongly depend on the selected execution mode (`main_1` or `main_2`), the simulation duration, and the numerical time step Δt. The approximate runtime on a standard 8–core CPU system and a RAM of 32 GB is summarized in Table 16. However, the table should be interpreted in conjunction with the following detailed discussion of each mode's limitations and recommended usage.

Table 16: Approximate simulation runtime and applicability.

Profile Length	Execution Mode	Typical Runtime
1 day (24 h)	`main_1` (single-pass)	1–2 minutes
1 year	`main_2` (chunked)	1–3 Hours

Main_1 Execution Mode

The `main_1` mode performs the entire electro–thermal simulation as a single unbroken process. This approach is suitable only for short mission profiles, typically up to one day (24 h) at a time resolution of $\Delta t = 0.001$ s. Simulations longer than four days quickly exhaust available RAM because all intermediate electrical, thermal and lifetime variables must be held in memory and plotted simultaneously.
Empirically, extending the duration beyond this limit results in excessive memory consumption (estimated at approximately 1.5 TB of RAM and over 1 TB of disk space for storing full DataFrames for a one-year mission). Consequently, `main_1` should only be used for short-duration simulations intended for model verification, debugging or sensitivity analysis.

Main_2 Execution Mode

The `main_2` mode was specifically developed to overcome the memory and data storage limitations of `main_1`. It implements a time *chunking* approach, dividing the simulation into segments of user-defined duration specified by the parameter `chunk_seconds`. For example, on a workstation equipped with 32 GB RAM, a chunk size of 86 400 steps (equivalent to one day of real time at $\Delta t = 0.001$ s) is computationally feasible. The recommended Δt values are either 0.001 s or 0.002 s. Larger steps degrade numerical accuracy, while smaller steps dramatically increase computation time.

For long-term simulations such as one year, only the final day's DataFrame (`df_1`) is saved to disk, as storing all intermediate daily data would require nearly 800 GB even with single-precision (`float32`) storage. Saving only the last day provides representative steady-state thermal and lifetime statistics without overwhelming disk capacity.

Optimization Strategy and Caching Mechanism

Both execution modes employ a fully vectorized electro–thermal computation framework. The electrical and reliability subroutines are array-based and parallelized wherever possible. However, the thermal rollout in `Electro_thermal_behavior_file.py` necessarily relies on `for`-loop iterations, as each RC branch temperature state depends on the previous time step. This serial dependency forms the principal computational bottleneck of the model. The overall speed can therefore only be improved by using a faster processor or optimized JIT compilation (Numba) which is used here.

To mitigate repetitive computation on the electrical side, a caching system is implemented using the `lru_cache` and `build_kernel_one_second_cached()` functions. Approximately 4 GB of RAM is allocated as cache memory. This cache stores precomputed electrical quantities (voltage, current, and power loss waveforms) for unique combinations of input parameters (P, Q, pf). When an identical set of input values reappears, the cached results are retrieved instantly without recalculating the electrical equations.

This mechanism is particularly effective when P, Q, and pf values remain constant for extended intervals (e.g., 15 minutes or 900 simulation steps). In such cases, the first step of the interval is calculated explicitly, and the following 899 steps are reconstructed directly from cached results, reducing runtime significantly.

For example, a one-year profile simulated in `main_2` with continuously varying P, Q, and pf can take up to three hours. Under a 15-minute resolution scenario (i.e., identical electrical values for 900 steps), the same case completes in approximately 50 minutes. However, this approach slightly overestimates lifetime predictions because it smooths short-term variations in thermal cycling that typically reduce device life. Similarly, a one-day simulation in `main_1` with 15-minute similarity can complete in about 20 seconds, whereas fully time-varying second-by-second profiles may require over one minute.

Recommended Practices

Users can balance simulation accuracy and computational performance by tuning three primary factors:

- **Chunking:** Divide long mission profiles into manageable daily segments when using `main_2`.
- **Caching:** Maintain an adequate cache size (4–8 GB) to reuse electrical kernel computations efficiently.
- **Input Resolution:** Design mission profiles such that P, Q, and pf values remain constant for short intervals (e.g., 15 minutes) to improve runtime without significantly compromising lifetime accuracy.

Finally, while higher CPU performance linearly improves execution speed, available RAM is the dominant limiting factor. Exhausting memory capacity will terminate the simulation prematurely. Therefore, prudent selection of chunk size, caching policy, and temporal resolution is critical for reliable and efficient long-term inverter lifetime simulations.

6 Model Validation and Sanity-Check

The present document is dedicated solely to describing the development, structure, and implementation of the electro–thermal lifetime assessment framework. Quantitative validation and detailed result interpretation are intentionally omitted here, as these will be comprehensively presented in a forthcoming research publication. The planned paper will include extensive simulation studies, comparison with experimental data, and statistical evaluation of the model's predictive accuracy under realistic inverter operating conditions.

6.1 Sanity Test Case and Expected Behaviour

As a preliminary internal verification step, a simple sanity test case was implemented to ensure numerical stability and physical consistency of the model. The scenario consists of a constant 34500 kW inverter output operating at unity power factor (pf = 1) and ambient temperature of 25 °C. Under these nominal conditions, both the IGBT and diode experience symmetrical conduction and switching losses, resulting in steady junction temperatures and lifetime degradation. This serves as the baseline case for comparing future test scenarios.
Subsequent sensitivity analyses performed with varying reactive power levels confirmed that the model responds in accordance with physical expectations:

- Increasing reactive power (deviation from unity power factor) results in higher RMS device currents and increased conduction losses.
- The corresponding rise in junction temperature swing (ΔT_j) accelerates thermal fatigue, thereby reducing the predicted lifetime.
- Lower power factor conditions produce asymmetric heating between the IGBT and diode, consistent with inverter current flow directionality.

These qualitative outcomes align with trends reported in the literature [2, 3], providing confidence in the correctness of the implemented electro–thermal coupling and lifetime estimation algorithms.

6.2 Planned Validation Work

A comprehensive validation campaign is currently under preparation and will be documented in a dedicated research paper. The forthcoming study will thus establish the empirical credibility of the proposed framework and quantify its accuracy across a range of inverter operating scenarios. For the present documentation, only the theoretical formulation and model implementation have been presented, while the complete experimental validation will follow in the subsequent publication.

Bibliography

[1] B. Bahrani, M. H. Ravanji, B. Kroposki, D. Ramasubramanian, X. Guillaud, T. Prevost, and N. Cutululis, "Grid-forming inverter-based resource research landscape: Understanding the key assets for renewable-rich power systems," *IEEE Power & Energy Magazine*, 2024. Date of current version: 22 February 2024.

[2] Y. Liu, L. M. Tolbert, P. Kritprajun, J. Dong, L. Zhu, J. Hambrick, K. P. Schneider, and K. Prabakar, "Aging effect analysis of pv inverter semiconductors for ancillary services support," *IEEE Open Journal of the Industrial Applications Society*, vol. 1, pp. 157–170, 2020. Invited Paper.

[3] U. Scheuermann, R. Schmidt, and P. Newman, "Power cycling testing with different load pulse durations," in *Proceedings of the International Conference on Power Electronics, Machines and Drives (PEMD)*, pp. 1–6, IEEE, 2014.

[4] A. Sangwongwanich, Y. Yang, D. Sera, and F. Blaabjerg, "Lifetime evaluation of grid-connected pv inverters considering panel degradation rates and installation sites," *IEEE Transactions on Power Electronics*, vol. 33, pp. 1225–1236, February 2018.

[5] W. Diao, J. Li, and G. Zhang, "A temperature-acceleration model for semiconductor device lifetime prediction based on the arrhenius equation," *Applied Sciences*, vol. 8, no. 12, p. 2402, 2018. Explains the derivation and application of Arrhenius-based acceleration factors for electronic component lifetime modeling.