# EMIoT Lab 1 Report:
# Dynamic Power Management (DPM) Simulation

Yuqi Li s336721  Xuxin Zhou s337564

Group 6

February 27, 2026

# Contents

# 1 Introduction

Dynamic Power Management (DPM) is a crucial low-power design technique... The goal of this lab is to use an event-driven C simulator to analyze and compare the energy performance of different DPM policies given a Power State Machine (PSM) and two distinct workloads (`workload_1.txt` and `workload_2.txt`).

## 1.1 Power State Machine (PSM) Analysis

The PSM used in this experiment consists of three states: RUN, IDLE, and SLEEP. Their power consumption and transition overheads are shown in the table below (data sourced from `psm.txt`):

Table 1: PSM Power Consumption and Transition Overheads

| **Parameter** | **Value** |
|---|---|
| $P_{\text{Run}}$ (Power) | 27.6000 mW |
| $P_{\text{Idle}}$ (Power) | 0.5700 mW |
| $P_{\text{Sleep}}$ (Power) | 0.0900 mW |
| RUN $\rightarrow$ IDLE (Energy / Time) | 0.0100 mJ / 0.4000 ms |
| IDLE $\rightarrow$ RUN (Energy / Time) | 0.0100 mJ / 0.4000 ms |
| RUN $\rightarrow$ SLEEP (Energy / Time) | 0.0200 mJ / 1.0000 ms |
| SLEEP $\rightarrow$ RUN (Energy / Time) | 2.0000 mJ / 4.0000 ms |

## 1.2 Break-even Time Calculation

To compare the IDLE and SLEEP policies, we must first calculate their break-even times ($T_{BE}$). $T_{BE}$ is the minimum idle duration required for the energy saved by entering the low-power state to offset the energy cost of the transitions exactly.
We calculate it using the formula:

$$T_{BE} = \frac{E_{\text{tr}} - (P_{\text{Idle/Sleep}} \times T_{\text{tr}})}{P_{\text{Run}} - P_{\text{Idle/Sleep}}}$$

where:

- $E_{\text{tr}} = E_{\text{Run}\rightarrow\text{Idle/Sleep}} + E_{\text{Idle/Sleep}\rightarrow\text{Run}}$
- $T_{\text{tr}} = T_{\text{Run}\rightarrow\text{Idle/Sleep}} + T_{\text{Idle/Sleep}\rightarrow\text{Run}}$

### 1.2.1 $T_{BE}$ (IDLE)

- $E_{\text{tr}} = 0.01 + 0.01 = 0.02$ mJ
- $T_{\text{tr}} = 0.4 + 0.4 = 0.8$ ms
- $T_{BE}$ (IDLE) $= \frac{0.02mJ - (0.57mW \times 0.8ms)}{27.6mW - 0.57mW} = \frac{0.019544}{27.03} \approx \mathbf{0.723}$ ms

### 1.2.2 $T_{BE}$ (SLEEP)

- $E_{\text{tr}} = 0.02 + 2.00 = 2.02$ mJ
- $T_{\text{tr}} = 1.0 + 4.0 = 5.0$ ms
- $T_{BE}$ (SLEEP) $= \frac{2.02mJ - (0.09mW \times 5.0ms)}{27.6mW - 0.09mW} = \frac{2.01955}{27.51} \approx \mathbf{73.41}$ ms

# 2 Part 1: Default Timeout Policy (RUN → IDLE)

Part 1 analyzes the **default timeout policy**, where the system transitions from the RUN state to the IDLE state when the inactive time $T_{\text{inactive}}$ exceeds the timeout value $T_{TO}$.

## 2.1 Workload 1 Analysis

We performed a sweep of $T_{TO}$ values for `workload_1.txt` (from 0ms to 200ms with a step of 1ms and from 0ms to 14ms). The figures follow:

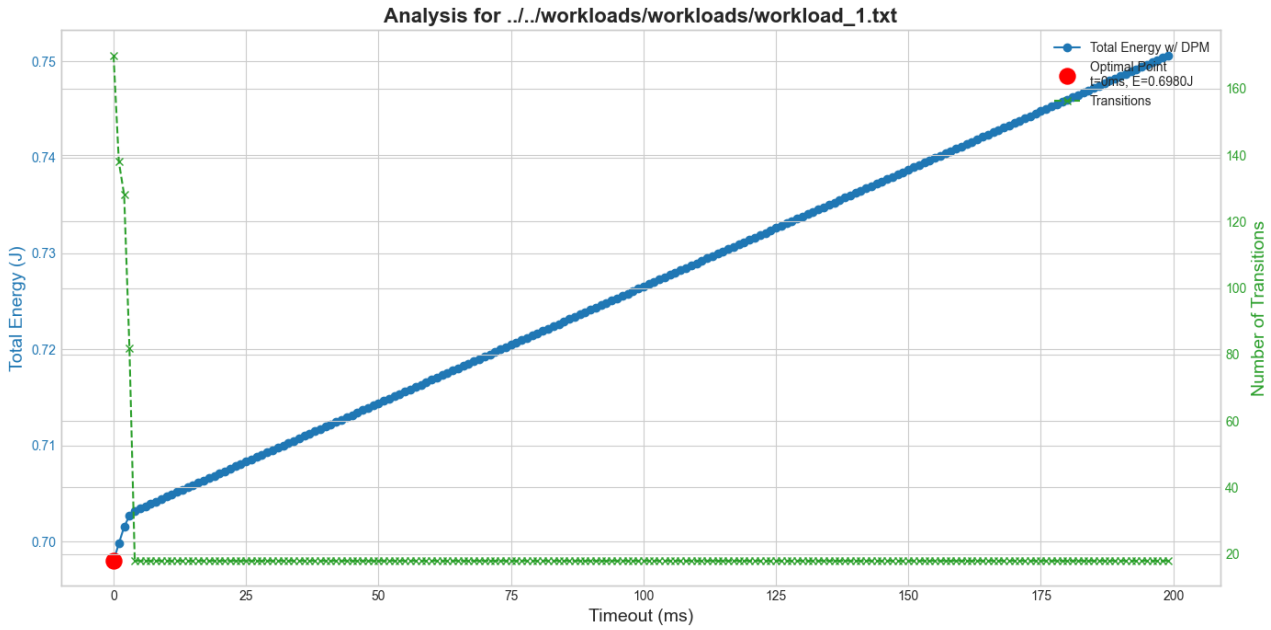


Figure 1: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 1 (RUN → IDLE).

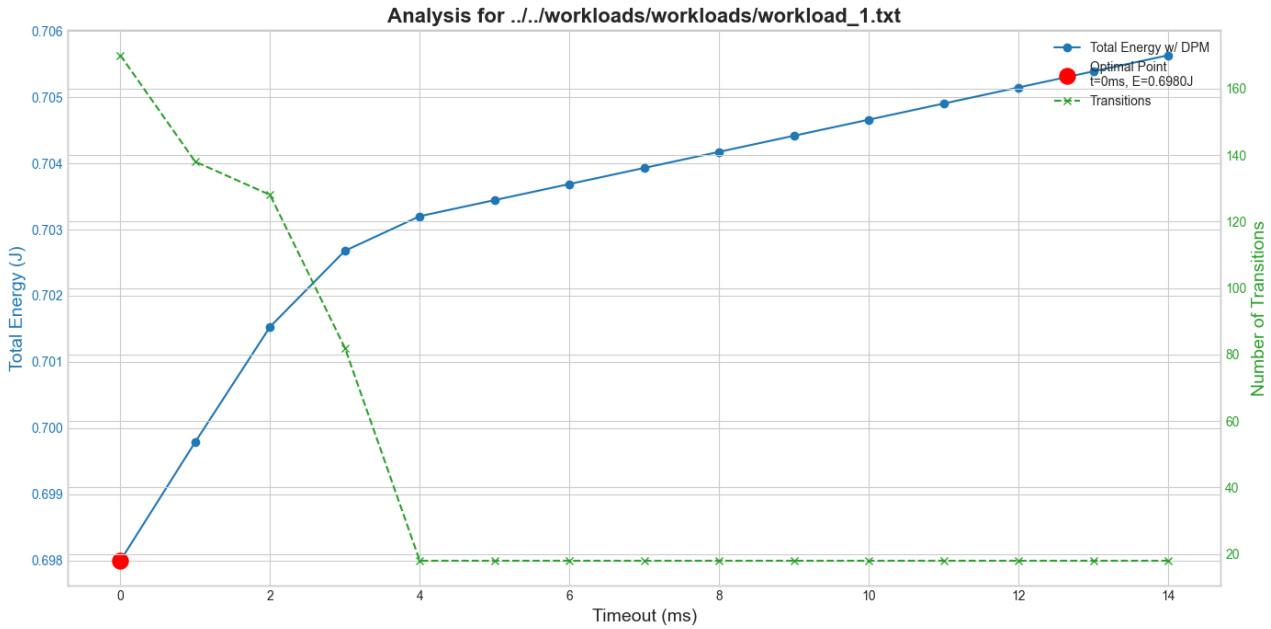


Figure 2: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 14ms for Workload 1 (RUN → IDLE).

### 2.1.1 Discussion

Previous figures show the **optimal point is 0ms**.
For part 1, it changes the state between IDLE and RUN, and the transition overhead is negligible. For this workload, it includes the majority of small idles like **3-4 ms** and some huge idles like **120000 ms.** Because the $Tbe = 0.723ms$, and even the shortest idle time (3-4 ms) in Workload 1 is much greater than this value, so the sooner you switch to IDLE state, the better.
When increasing the timeout value, the result can be divided to 2 parts.

- **The first part** is 0-4 ms timeout range. When the timeout is low (e.g., 0ms), the system immediately switches to the power-saving Idle state; although switching is frequent, the system spends most of this time saving power. As the timeout increases toward 4ms, the system stays in the high-power Active state to cover these gaps. While this reduces state transitions, maintaining the Active state consumes far more energy than switching saves, causing total energy usage to spike.

- **The second part** is above 4 ms. Short gaps are now fully covered by the Active state, leaving only rare long idle periods (larger 120,000ms). For these long breaks, the system waits in the high-power Active state for the full timeout duration before switching to Idle. Every 1ms added to the timeout adds 1ms of high-power waiting time to the start of each long break. Since the number of long breaks is fixed, this wasted energy accumulates, leading to a steady linear increase in consumption.

## 2.2 Workload 2 Analysis

We performed a $T_{TO}$ sweep for `workload_2.txt` (e.g., from 0ms to 200ms with a step of 1ms). The figure follows:

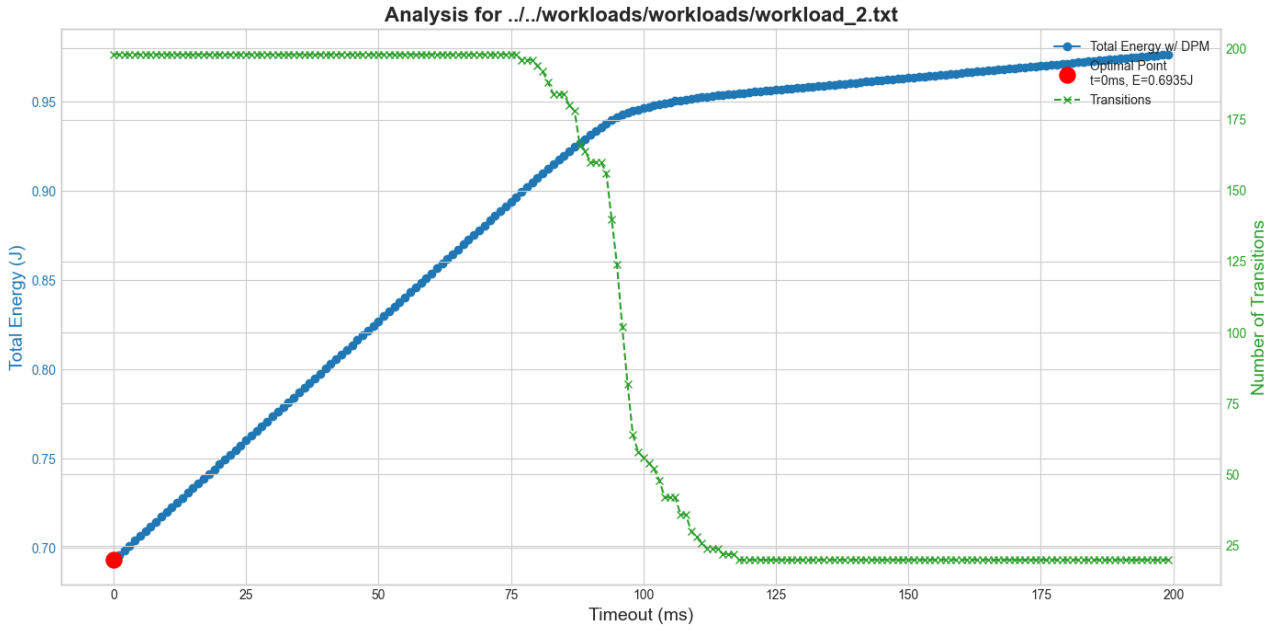


Figure 3: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 2 (RUN $\rightarrow$ IDLE).

### 2.2.1 Discussion

The previous figure shows that the **optimal point is 0ms**.

Workload Characteristics: Workload 2 includes two types of idle periods:

- **Majority:** Short idle periods distributed continuously between **75 ms and 120 ms**.
- **Minority:** Rare but extremely long idle periods (approximately **120,000 ms**).

In Workload 2, because its shortest free time (about $75ms$) is also more than 100 times larger than $T_{BE}$(0.723 ms), the sooner it enters the idle state, the better, and workload 1 is the same. The difference from workload 1 is that the energy curve of Workload 2 will have an arc area between $75-120ms$. Workload 1 forms an arc area around $3-4ms$, because with the increase of $T_{TO}$, more and more idle periods are covered by the RUN state, and the conversion overhead is omitted, but the total energy consumption is still much higher than the immediate switching.

To explain the shape of the energy curve in Figure 3, we analyze the energy cost for a single idle period of duration $D$ given a timeout $T_{to}$. The behavior leads to two distinct cases:

**Case A: Uncovered ($T_{to} < D$)**

When the timeout is shorter than the Majority idle gap, the system waits in the high-power Run state for the duration of $T_{to}$ and then switches to the low-power Idle state. We model the energy consumption as follows:

$$E_{uncovered} = (P_{run} \cdot T_{to}) + E_{trans} + P_{idle} \cdot (D - T_{to}) \tag{1}$$

In this case, as $T_{to}$ increases, the total energy rises rapidly because we are replacing low-power Idle time with high-power Run time.

**Case B: Covered ($T_{to} \geq D$)**

When the timeout is longer than the idle gap, the system remains in the Run state for the entire duration $D$. No state transition occurs. We model the energy consumption as follows:

$$E_{covered} = P_{run} \cdot D \tag{2}$$

In this case, the energy consumption becomes constant. It does not increase with further extensions of $T_{to}$. Additionally, the transition cost ($E_{trans}$) is eliminated.
Based on the mathematical model, the energy curve trend can be explained in three phases:

- **Phase 1: Linear Increase ($<$ 75 ms)**
  All short idle periods fall into **Case A**. Increasing the timeout by 1 ms forces every idle period to consume more energy ($P_{run} \cdot 1$ms). Consequently, the total energy rises sharply with a steep slope.
- **Phase 2: The Gradual Slowing (75 ms $-$ 120 ms)**
  This region exhibits a curved shape rather than a sharp turning point. This is due to the distribution of short idles between 75 ms and 120 ms. As $T_{to}$ increases, it gradually covers the short idle periods one by one. When an idle period shifts from Case A to Case B:

  1. Its energy cost becomes capped at $P_{run} \cdot D$ and stops contributing to the slope of the curve.
  2. The transition overhead ($E_{trans}$) is saved, further reducing the total energy slightly.

  As fewer idle periods remain "uncovered" to drive the energy increase, the slope of the curve gradually flattens, creating the observed arc.

- **Phase 3: Linear Stability ($> 120$ ms)**
  At this stage, all short idle periods are fully covered (Case B) and consume constant energy. The slow linear increase in total energy is driven solely by the few remaining long idle periods (approx. 120,000 ms), which remain in Case A.

# 3 Part 2: Extended Timeout Policy (RUN $\rightarrow$ SLEEP)

Part 2 modifies the `dpm_decide_state` function in `dpm_policies.c` to change the target state of the timeout policy from IDLE to SLEEP.

## 3.1 Analysis: IDLE vs SLEEP

We re-ran the $T_{TO}$ sweeps from Part 1 using the modified simulator (RUN $\rightarrow$ SLEEP).

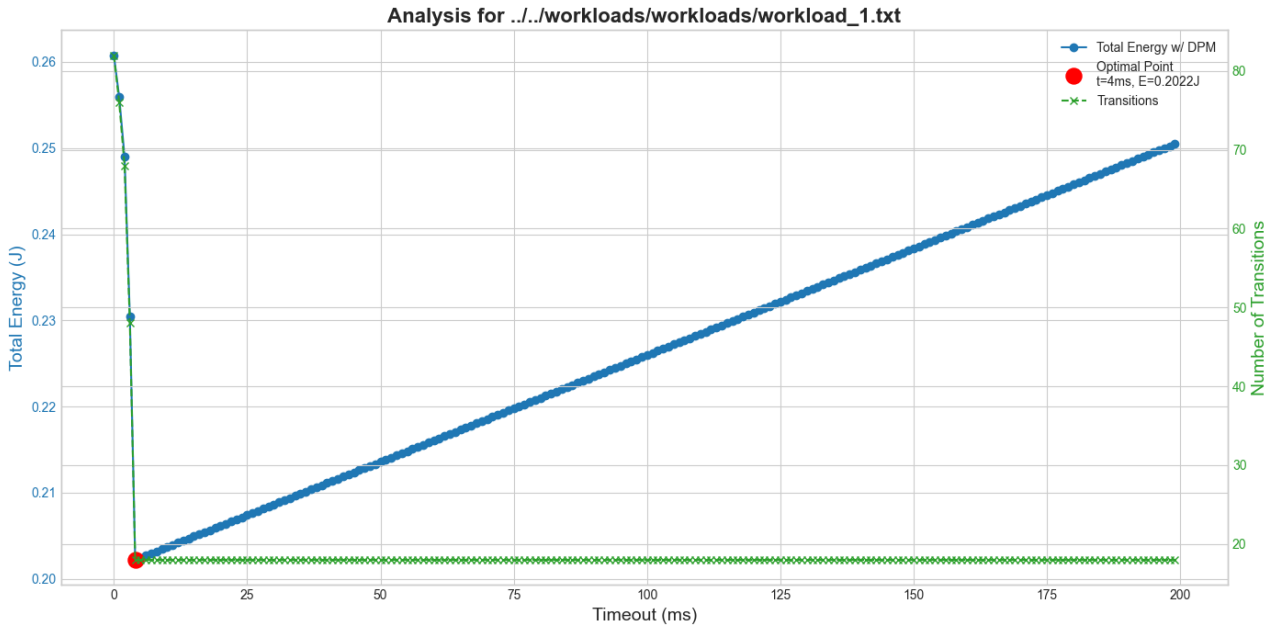


Figure 4: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 1 (RUN $\rightarrow$ SLEEP).

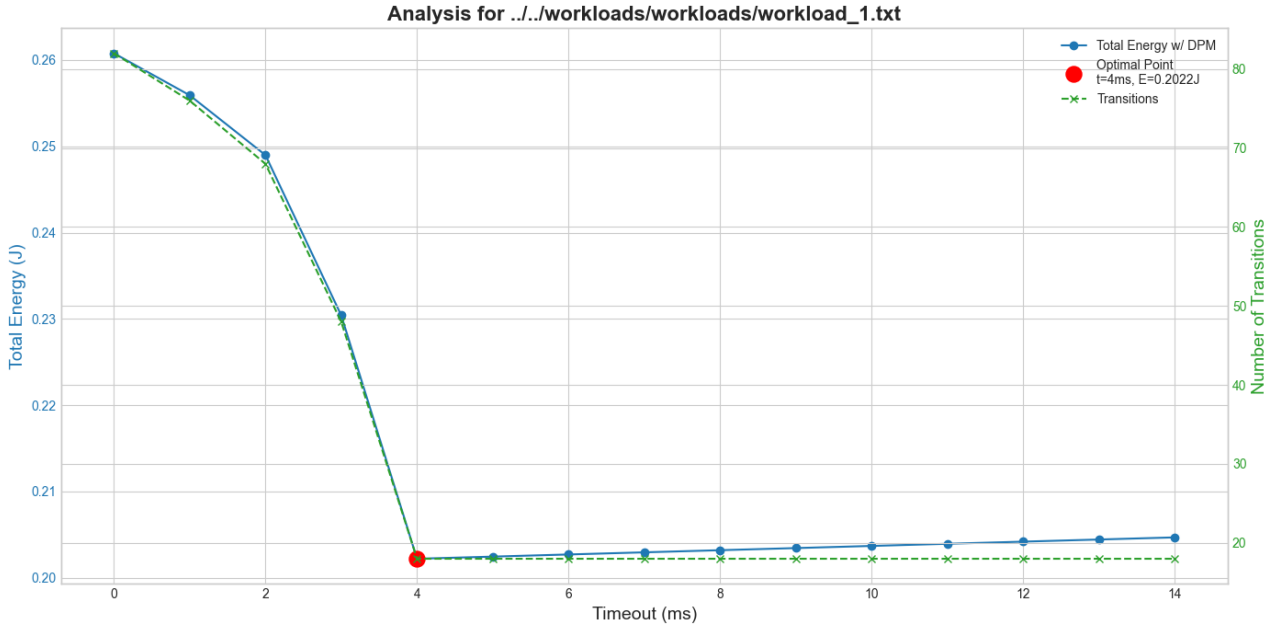


Figure 5: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 14ms for Workload 1 (RUN $\rightarrow$ SLEEP).

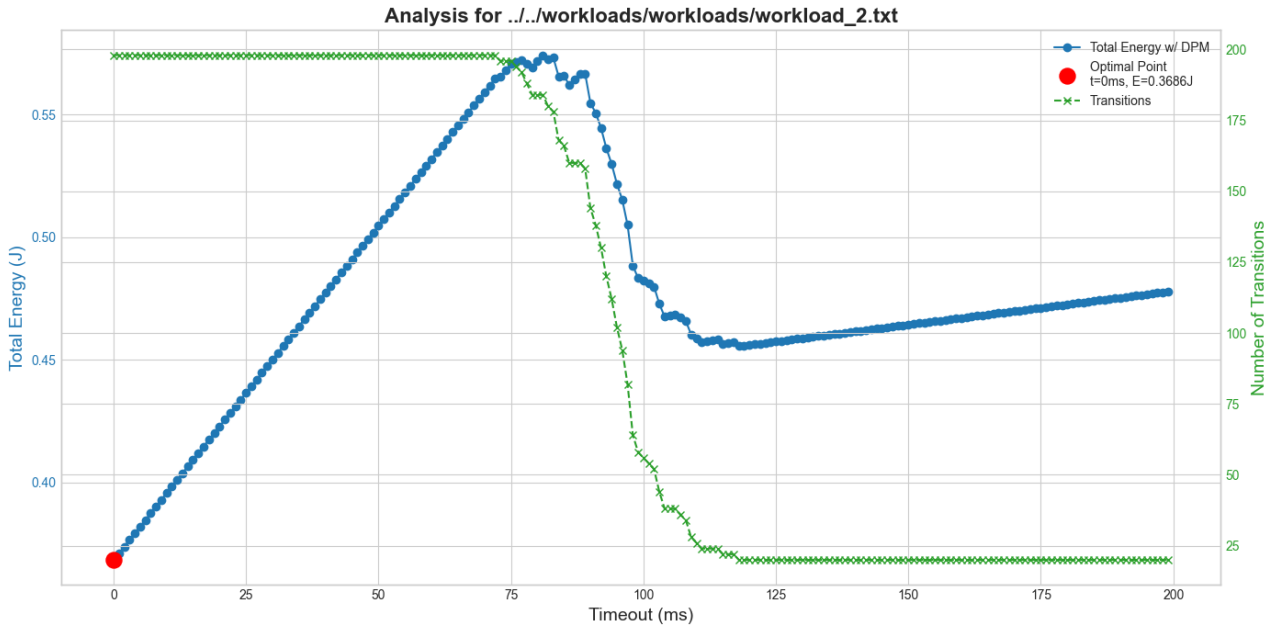


Figure 6: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 2 (RUN → SLEEP).

### 3.2 Discussion (Answering Part 2 Questions)

Comparison between RUN → IDLE timeout policy and RUN → SLEEP timeout policy.

**1. What changes?**
The optimal point changed, because of the transition cost different, and the different break-even time, in RUN → IDLE has a low transition cost and short break-even time. In RUN → SLEEP, the high transition cost and longer break-even time, and stay in the SLEEP state can save more energy than IDLE state.

**2. What are the best $T_{TO}$ value in the two cases?**
For workload 1, in the RUN → IDLE policy, the break-even time is really small, the IDLE times are higher than that, so it needs to transition immediately otherwise will lead to an energy penalty(energy cost in RUN time waiting). For RUN → SLEEP policy, the best $T_{TO}$ is 4ms, in 4ms, the energy cost decline to the lowest, at this point, the 4ms timeout represents a filter, filter out the frequent short IDLE times(less than 4ms), and allow system to sleep in long IDLE times(120s).
For workload 2, both for this two timeout policy, the best $T_{TO}$ is 0ms, the best Strategy is immediate transition, because the IDLE time always longer than break-even time, any timeout will lead to the energy penalty.

**3. Which of the two results in the overall lower energy?What changes for the two workloads provided?**
The RUN → SLEEP result in the lower overall energy, because the energy cost for stay in the SLEEP state(0.09mW) is much lower than stay in IDLE state(0.57mW). For workload 1, Change: The strategy must be switched from "Immediate Switch" to "Filter Mode".Because the WL1 contains many IDLE times that less than the break-even time, if the timeout value is set to 0ms like IDLE policy, it will need to much energy penalty.Thus should be use 4ms to filter out the short IDLE times, only transition for long IDLE period.
For workload 2, for the two policies the optimal point is the same, because all the idle periods are higher than the two break-even times.So it should remain the immediate transition strategy to get the optimal point. But the break-even times of two policies is different, the profit window for WL2 is changed. For the RUN → IDLE policy, most of IDLE time is much higher than it's break-even

time(0.7ms),means that a longer timeout (100ms - 0.7ms =99.3ms)is acceptable to get the profit although it's not the optimal point, but for RUN → SLEEP policy, the most IDLE time is little high than the break-even time(75ms),lead to shorter timeout value to get the profit.

**4. Why?**

The analysis below answer the Why in detail.

## 3.3 Analysis of Timeout Policy to SLEEP

In Part 2, we modified the timeout policy to transition into the **SLEEP** state instead of IDLE. The results exhibit distinct behaviors for the two workloads due to the high transition cost of the SLEEP state ($E_{tr} \approx 2.02$ mJ) and its high break-even time ($T_{be,sleep} \approx 75$ ms).

### 3.3.1 Analysis of Workload 1

Figures 4 and 5 illustrate the energy consumption and transition counts for Workload 1. The optimal point is observed at $T_{to} = 4$ms.

- **Phase 1: The Energy penalty of Misprediction ($T_{to} < 4$ms).** For timeout values less than 4ms, energy consumption is high ($\approx 0.26$ J). Workload 1 contains frequent short idle gaps of exactly 4ms. When $T_{to} < 4$ms (less than most of idle gaps), the system triggers transition to SLEEP during these short gaps. But the gap duration (4ms) is significantly shorter than the break-even time (75ms), the energy saved in SLEEP fails to cover the high transition cost, resulting in energy loss.

  As the timeout value increases, it effectively 'covers' the smaller idle gaps (like 1ms, 2ms, and 3ms) in the workload. This prevents the system from switching to SLEEP during these short periods, avoiding the high energy penalty of the transition. At the same time, this reduces the total number of transitions, leading to a gradual decrease in power consumption.

  Notes: Here, the number of transitions reduced can lead to energy drop is because during this phase, the energy saving of reduced transitions is much higher than energy penalty of increased waiting time.

  **Equivalent Cost:**

$$\frac{E_{tr}}{P_{Run}} = \frac{2.02 \text{ mJ}}{27.6 \text{ mW}} \approx \mathbf{73.19} \text{ ms}$$

  - **Meaning:** In this system, the energy consumed by a single switch is equivalent to letting the CPU run for 73ms.

- **Phase 2: The Optimal Point ($T_{to} = 4$ms).** At exactly $T_{to} = 4$ms, energy consumption drops sharply to its minimum ($\approx 0.20$ J). Here, the timeout acts as a filter. The system waits in the RUN state for 4ms; since the short idle periods end exactly at 4ms, the system detects the new task and cancels the transition to SLEEP. This prevents the expensive transition costs for short gaps while still allowing transitions during the rare long idle periods (120s, larger than break-even time of sleep), capturing maximum savings.

- **Phase 3: Linear increase ($T_{to} > 4$ms).** Since the timeout value is larger than the short gaps (4ms), the system stays in RUN during these short breaks. For the long 120s gaps, the system must wait for the timeout before switching to SLEEP.

  As the timeout value increases, the system spends more time 'waiting' in the high-power RUN state. This means there is less time left for SLEEP within the 120s interval. Essentially, expensive running time is replacing cheap sleeping time, causing the total energy to increase linearly.

### 3.3.2 Analysis of Workload 2

Figure 6 shows the results for Workload 2, which consists of regular, long idle periods of approximately 100ms.

- **Phase 1: The Waiting Penalty ($0 < T_{to} < 75$ms).** The optimal strategy is immediate transition ($T_{to} = 0$) with an energy of $\approx 0.37$ J. As $T_{to}$ is less than the most of idle duration, in this phase the system is switched to SLEEP state, As $T_{to}$ increases, the system wastes energy waiting in the RUN state before transitioning to SLEEP, Thus the energy cost increased linearly. The most of idle duration (100ms) is always longer than $T_{be,sleep}$ (75ms), switching to the SLEEP can save energy, the main problem of energy consumption increase is the increase waiting time in RUN state.

- **Phase 2: Energy drop ($75 < T_{to} < 120$ms).** In this phase, the total energy consumption drops. This happens because a longer timeout prevents the system from making useless, expensive state transitions at the very end of an idle period. And most idle duration are approximately 100ms. Let's use this value as an example for analysis. Based on the logs we exported, we selected two time points for analysis.
  **The Worst-Case Scenario ($T_{to} = 90$ms):** The system waits 90ms in the RUN state and then triggers the timeout to enter SLEEP. However, the next task arrives just 10ms later ($100 - 90 = 10$). The system is forced to wake up almost immediately. It pays a massive transition cost but saves practically no energy. This results in 144 transitions and a high energy cost of 0.554 J.
  **Avoiding the Transition ($T_{to} = 110$ms):** The system waits in RUN. The new task arrives at 100ms, which is before the 110ms timeout. As a result, the timeout never triggers. The system simply stays in RUN and completely avoids the expensive SLEEP transition. The number of transitions drops sharply to 26, and the total energy drops to 0.458 J.

  Note: The cost of transitions for the SLEEP state is much higher than IDLE state. That's why there is a drop but for the IDLE policy didn't have.

- **Phase 3: Linear Stability ($T_{to} > 120$ms).** In this Phase, most of the idle duration time is less than $T_{to}$, for these durations, the system will not switch to the SLEEP state. But there still exist some large idle durations(approx. 120,000 ms), for them the system will switch to the SLEEP state, but the $T_{to}$ increase, the system needs the more time to wait before switching to the SLEEP state. Thus, the energy consumption increase.

**Conclusion:** The transition to SLEEP requires a careful timeout selection. For workload 1, the timeout must act as a **filter** (4ms) to avoid break-even penalties. For workload 2, the timeout should be minimized (0ms) to avoid waiting penalties.

# 4 Part 3: Predictive Policy

## 4.1 Policy Summary

We implemented 2 prediction strategies.

- **Simple Predictive Policy**: It uses the predictive logic with the $T_{be}$ setting to decide whether to switch to SLEEP or IDLE state.

- **Predictive Hybrid Policy**: It combines predictive logic with the $T_{to}$, where the thresholds are tuned to the dominant idle times of the two workloads.

We adopted a simplified **Last Value Prediction** strategy. For the provided workload 1 and workload 2, the IDLE periods are repetitive, and the duration of the current inactive period is likely identical

to the previous one.
We utilized the **Break-even Time** calculated in the previous section to guide state transitions (**$T_{be,sleep} \approx 75$ms and $T_{be,idle} \approx 0.8$ms**) for Simple Predictive Policy.
We based on the dominant idle times of the two workloads to set the **safe** $T_{to}$ for Predictive Hybrid Policy.
The policy logic was detailed as follows:

### 4.1.1 Simple Predictive Policy

**Logic:**

- If $T_{predicted} \geq T_{be,sleep} \rightarrow$ Transition to **SLEEP** immediately.
- Else if $T_{predicted} \geq T_{be,idle} \rightarrow$ Transition to **IDLE** immediately.
- Otherwise, stay in **RUN**.

**Outcome:**
The strategy works very well on Workload 2 because its idle periods are long and predictable. Most idle times are longer than $T_{be,sleep}$. Even when the idle time drops from 120s to 100ms, entering SLEEP is still profitable.
However, the strategy fails on Workload 1. Most idle periods in Workload 1 are just 4ms (less than $T_{be,sleep}$). Critically, the workload often jumps from a long idle (120s) directly to a short one (4ms). This tricks the system into thinking it should sleep. Since 4ms is far shorter than the break-even time, this leads to massive energy waste.

### 4.1.2 Predictive Hybrid Policy

**Logic:**

- If $T_{predicted} \geq T_{be,sleep} \rightarrow$ Transition to **SLEEP** immediately.
- Else $\rightarrow$ Remain in **RUN** for a safety timeout ($T_{to} = 4$ms).
  - If idle time exceeds 4ms $\rightarrow$ Transition to **SLEEP**.
  - Otherwise $\rightarrow$ Remain in **RUN**.

**Outcome:**
This strategy significantly improves energy efficiency on Workload 1 by leveraging prior knowledge from Part 2. By using a 4ms safety timeout, it solves the failure case of the previous policy: when the workload shifts from a short idle to a long one, the timeout mechanism catches the transition and forces the system to sleep, avoiding the massive waste of remaining in RUN.
However, the policy still has limitations. When transitioning from a long idle (120s) to a short one (4ms), the history-based prediction incorrectly triggers an immediate SLEEP, causing an energy penalty. This policy represents a case of overfitting. The 4ms threshold was manually selected based on specific workload analysis rather than runtime adaptation. If the noise floor were to increase (e.g., to 5ms), this hard-coded filter would fail, making the policy unstable in unknown environments.

## 4.2 Results and Comparison

The table below summarizes the best energy results achieved by each policy.

Table 2: Total Energy Consumption (J) by Policy

| **Policy** | **WorkLoad 1** | **WorkLoad 2** |
|---|---|---|
| *Baselines (Part 1 and 2)* | | |
| P1: Timeout → Idle | 0.698 J | 0.693 J |
| P2: Timeout → Sleep | **0.202 J** | 0.379 J |
| P2: Timeout → Sleep | 0.260 J | **0.369 J** |
| *Custom Policies (Part 3)* | | |
| 1. Simple Predictive | 3.957 J | 0.392 J |
| 2. Predictive Hybrid | **0.219 J** | **0.369 J** |

## 4.3 Discussion

### 4.3.1 Comparison between predictive policy and timeout policies: What changes?

Comparing the baseline Timeout policies (Part 1 & 2) with the Predictive policies (Part 3), the fundamental change is the shift from a **reactive** to a **proactive** decision-making mechanism, eventually evolving into a hybrid approach.
**Why:**

- **Timeout Policies (Part1 & Part2):** These are purely reactive mechanisms. The system must waste energy waiting in the RUN state for a fixed duration ($T_{to}$)—which is **manually configured by the user**—before transitioning.
  - *P1 (Timeout → Idle):* Safe but inefficient ($\approx$ 0.7 J) as it relies on the user setting a timeout without leveraging the low power SLEEP state.
  - *P2 (Timeout → Sleep):* More efficient than switch to IDLE state in most timeout cases. It acts as a static filter. Only after the user manually tunes $T_{to}$ to the optimal value (4ms), does it achieve the minimum energy.
- **Predictive Policies (Part 3):** These attempt to transition immediately ($T = 0$) based on historical trends.
  - *Simple Predictive:* Failed on Workload 1 due to the high cost of mispredictions.
  - *Predictive Hybrid:* Combined the best of both policies(predictive and static timeout). They utilize prediction for zero-latency sleep during long idle periods (if the predict the long idle period, go to sleep with no waiting) and introduce a safety timeout to filter out mispredictions on Workload 1(the case of from short idle period into the long idle period).

### 4.3.2 Which approach is the best for the two workloads?

It depends on the different workloads, Based on the results in Table 2, For the workload 1 the best approach is Timeout policy (switch to sleep) in 4ms $T_{to}$ value, and for workload 2, the best approach is Predictive Hybrid , although the Timeout policy (switch to sleep) in 0ms $T_{to}$ value can also achieve the lowest energy, but it should to manually set the different timeout value to find the optimal point.
**Why:**

- **For Workload 1:** Based on the result, the best strategy is the timeout policy at 4ms(switch to Sleep), because the 4ms timeout value can filter the small idle periods and prevent the energy penalty. and for the Predictive Hybrid policy (0.219 J) is consumes slightly more than the theoretical limit of previous strategy (0.202 J) because of when this policy meet the condition of from long idle period to short period, the misprediction will lead to the energy penalty. and both of the previous policies are better than the Simple Predictive policy (3.957 J) and is significantly

better than the P1 baseline (0.698 J), because stay on the IDLE state could take more energy than SLEEP state by definition of the PSM.

- **For Workload 2:** The **Predictive Hybrid (0.369 J)** ties with P2 for the best performance. It is superior to P1 (0.693 J) by approximately **49%**, achieved by correctly predicting long idle durations and switching to SLEEP immediately.

#### 4.3.3 Why does a predictive approach work better/worse?

The performance depends on the **Different distribution of workload** of the workload and the **cost of misprediction**.

- **Why Simple Prediction failed on Workload 1:** The result of TABLE shows that there is a big penalty(3.957J) far greater than other results, by checking the runtime log, found that there is a huge time spend on the RUN state(122.84s), and we check the original workload file, we found there is no situation meets the condition of previous IDLE period less than the break-even time of IDLE AND the next IDLE period is 120s (only in this situation can lead the simple prediction let the 120s long period in RUN state). To finding the real situation handled by PSM, we check the PSM code, and by adding the printf function after **dpm_update_history** statement the in **dpm_simulate** function in **dpm_policy.c**, we found that the real IDLE period stored in the history array is not equal to the ideal IDLE period shown in workload file, it's equal to the Next arrival time - Current end time, it depends on the real-time PSM decision and consider the transition costs. Thus, there is a original 4ms IDLE happened in time point at 841936 in workload file, by effect of the previous misprediction penalty of transition cost, this IDLE period is reduced to less than the break-even time of IDLE state, thus let the next IDLE period(120s) keep in RUN state, lead to huge energy waste.
- **Why Hybrid Prediction succeeds on Workload 2:** Workload 2 is highly periodic with stable idle durations ($\approx$ 100 ms). Here, prediction is extremely accurate. Since the idle duration consistently exceeds the break-even time, the predictive policy correctly decides to SLEEP immediately. Unlike timeout policies that waste energy waiting for the user-defined $T_{to}$, the predictive policy achieves **zero-latency switching** to other states.

## 5 Extra Challenge

### 5.1 Try other policies

#### 5.1.1 Adaptive Hybrid Policy

**Logic:**
The policy dynamically calculates a noise floor based on the history window $H$ to set an adaptive safety timeout ($T_{safety}$), rather than using a hard-coded value.

- **Dynamic Parameter Calculation:** At each decision point, the system identifies the maximum duration of recent "short" idle periods (noises):

$$T_{safety} = \max(\{t \in H \mid t < T_{be,sleep}\}) \tag{3}$$

  This formula selects the worst-case recent noise. The rationale is asymmetric: setting the timeout slightly too long only wastes a few milliseconds of RUN power, but setting it too short causes an expensive transition to SLEEP during a short burst (false positive).

- **Decision Rule:**
  - If $T_{predicted} \geq T_{be,sleep} \rightarrow$ Transition to **SLEEP** immediately.

- Else → Wait for $T_{safety}$.
  - If active before $T_{safety}$ → Stay in **RUN** (Filtered as noise).
  - If inactive after $T_{safety}$ → Transition to **SLEEP** (Confirmed long idle).

**Outcome:**
This adaptive approach successfully solves the generalization problem. Without any manual tuning, the algorithm automatically converged to a safety timeout of ≈ 4.1ms for Workload 1 and ≈ 0.5ms for Workload 2. By using the historical maximum noise as the threshold, the system effectively distinguishes between short bursts and actual long idle periods. Although this method consumed marginally more energy (+0.005 J) than the manually tuned version (due to the learning phase and safety margins), it achieves robust performance across unknown environments, eliminating the risk of overfitting associated with fixed-parameter policies.

## 5.2 Change the PSM

By modifying the psm.txt file, we change the PSM parameters into the new state below:

Table 3: New PSM Power Consumption and Transition Overheads

| **Parameter** | **Value** |
|---|---|
| $P_{\text{Run}}$ (Power) | 150.0000 mW |
| $P_{\text{Idle}}$ (Power) | 30.0000 mW |
| $P_{\text{Sleep}}$ (Power) | 10.0000 mW |
| RUN → IDLE (Energy / Time) | 0.0500 mJ / 0.1000 ms |
| IDLE → RUN (Energy / Time) | 0.0500 mJ / 0.1000 ms |
| RUN → SLEEP (Energy / Time) | 0.1000 mJ / 0.2000 ms |
| SLEEP → RUN (Energy / Time) | 0.5000 mJ / 0.5000 ms |

And we re-run the same time-out policy experiment same as the previous Part1 and Part2 for both two workloads.

### 5.2.1 Break-even Times

$T_{BE}$ (IDLE)

- $E_{\text{tr}} = 0.05 + 0.05 = 0.1$ mJ
- $T_{\text{tr}} = 0.1 + 0.1 = 0.2$ ms
- $P_{\text{tr}} = 0.1mJ/0.2ms = 500$ mW
- $T_{BE}(\text{IDLE}) = \frac{0.1mJ-(30mW\times0.2ms)}{150.0mW-30mW} \approx \mathbf{0.783}$ ms

$T_{BE}$ (SLEEP)

- $E_{\text{tr}} = 0.1 + 0.5 = 0.6$ mJ
- $T_{\text{tr}} = 0.2 + 0.5 = 0.7$ ms
- $P_{\text{tr}} = 0.6mJ/0.7ms = 857$ mW
- $T_{BE}(\text{SLEEP}) = \frac{0.6mJ-(10mW\times0.7ms)}{150.0mW-10mW} \approx \mathbf{4.236}$ ms

### 5.2.2 The policy for RUN to IDLE

The result of this PSM for the IDLE timeout policy is shown in Figure 7 and Figure 8,the curve trend is quite similar with the previous results of the original PSM for RUN to IDLE timeout policy. Because the features of workloads didn't change, and the all the IDLE period is higher than the break-even time of this IDLE policy.
But the differences is that the overall energy consumption of the new PSM provide for this policy is much higher than the previous PSM(32J vs 0.7J). Because the energy consumption of the New PSM of each state is much higher than the original PSM and the transition overheads of the IDLE state is also higher than original one.

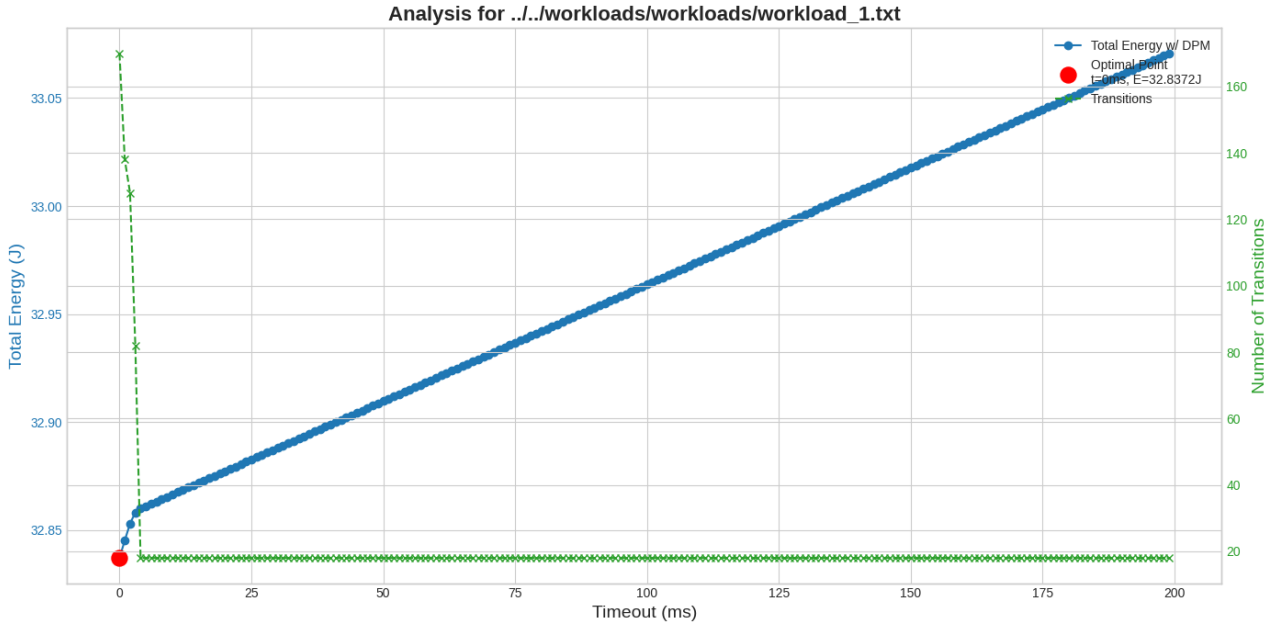


Figure 7: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 1 (RUN $\rightarrow$ IDLE).

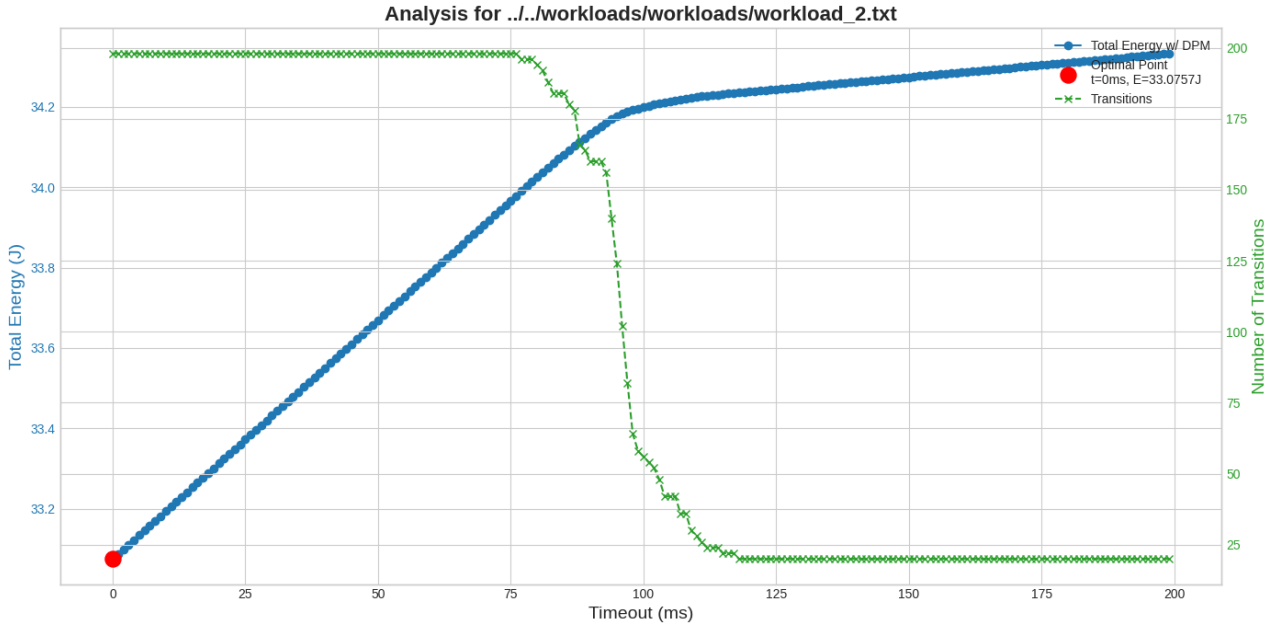


Figure 8: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 2 (RUN $\rightarrow$ IDLE).

### 5.2.3 The policy for RUN to SLEEP

The result of this PSM for the SLEEP timeout policy is shown in Figure 9 and Figure 10. Compare to the original PSM, the break-even time of new PSM for the SLEEP state is much lower(4.236ms vs 75ms). There is a difference between new PSM for workload1 and the previous(Figure4). A slightly increase and drop happens before the timeout value reach the optimal point. That's because at the begin of timeout value increase, costs for waiting in RUN state increased, lead to energy penalty(by analyze the log and figure, we found that from the time-out value 0 to time-out value 2, the energy

cost move from 11.286J to 11.295J, during this period, the waiting-time increased, and the number of transitions decreased from 170 to 128, the energy saving rough at 0.0126 J, but the energy penalty for the waiting time is much higher the energy saving for reduced transitions, thus total energy cost increased. For the inflection point, for the time-out value from 2 to 4, the energy cost is decreased, because the number of transitions dropped from 128 to 18(lots of task with the short period covered by time-out value,system not try to go to sleep), and the energy penalty of waiting time is much lower than the energy saving of reduced transitions, thus the energy dropped to the optimal point. After this point, the timeout filter the IDLE periods(4ms or less), keep in RUN state for that periods(huge energy cost in RUN), and allow switch to the SLEEP for 120s period, as the timeout increase, waiting time in RUN before switch to SLEEP increase, thus leading to the linear increase.
The difference for the result of workload2 compared to the original PSM achieved(Figure 6) is that when the timeout value is in the range of from 75ms-100ms, although the number of transitions dropped, but the energy consumption did not drop. Because compared to the overhead of staying in SLEEP state or RUN, the magnitude of the switching overhead is negligible. But in the previous result in figure 6, the overhead of transitions is not negligible compared to the overhead of stay in SLEEP or RUN state.

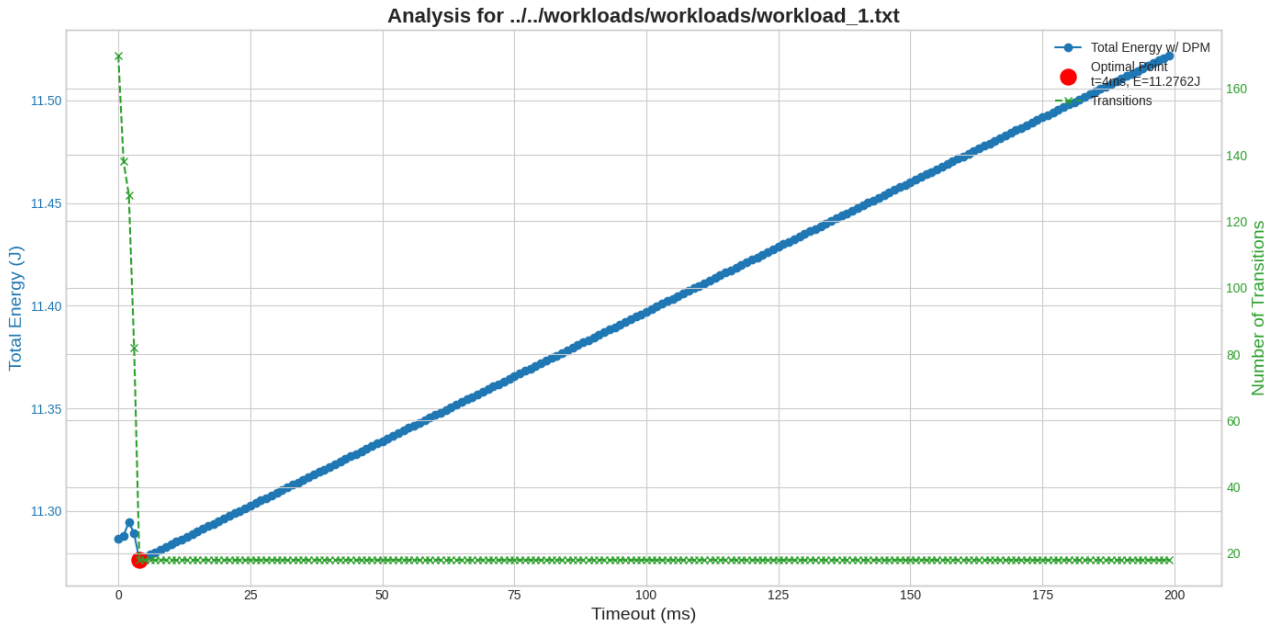


Figure 9: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 1 (RUN $\rightarrow$ SLEEP).

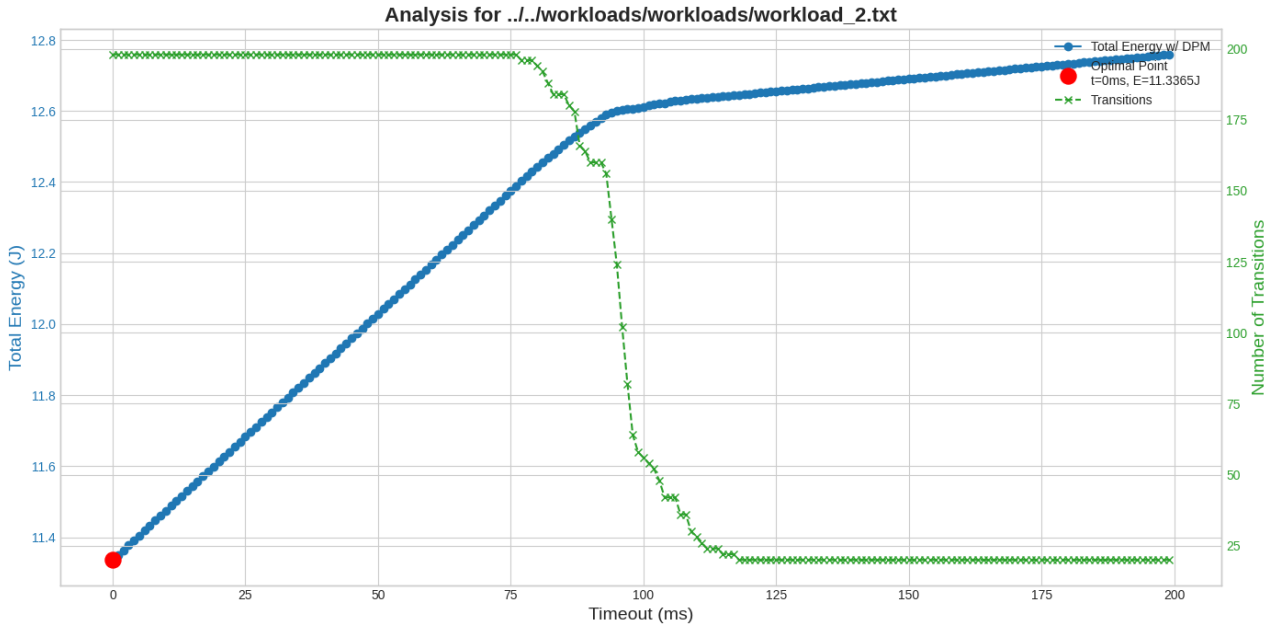


Figure 10: Energy consumption and number of transitions vs. $T_{TO}$ 0ms to 200ms for Workload 2 (RUN $\rightarrow$ SLEEP).

## 5.3 Change the workload

We have added two additional workloads to test the implementation of different strategies under different workloads.

### 5.3.1 Hover Workload

Long(120s) and short IDLE time(4ms) alternately appear. To test the prediction algorithm's performance during sudden state changes.

### 5.3.2 Ramp Workload

Short noise bursts (6ms) and long idle periods (100s) appear cyclically. To distinguish between the Predictive Hybrid Policy and the Adaptive Hybrid Policy by testing the system's ability to filter out high-amplitude noise.

## 5.4 Result and Comparison

In this part, we tested **3 history strategies**, and got different behaviors and results in **two new workloads**.

Table 4: Total Energy Consumption (J) by Policy (Hover vs. Ramp Scenarios)

| **Policy** | **Hover Workload** | **Ramp Workload** |
|---|---|---|
| Simple Predictive | 0.7238 J | 1.2019 J |
| Predictive Hybrid (Fixed) | 0.6709 J | 0.3200 J |
| Adaptive Hybrid (Advanced) | 0.6708 J | 0.2836 J |

## 5.5 Discussion

### 5.5.1 Hover

The Hover workload alternates between long idle periods (120s) and short idle periods (4ms). Its goal is to test how the policy handles non-profitable short idles and to highlight the flaws of the Simple Predictive Policy.
The results show that the Simple Policy had the highest energy consumption (0.7238 J). This is because it relied solely on history and incorrectly switched to sleep immediately when the 4ms idle began. Since 4ms is far below the break-even point (75ms), the system suffered from unnecessary wake-up penalties.
In contrast, the Hybrid strategies (Fixed and advanced) achieved better efficiency (0.6709 J) by using a timeout mechanism. When facing the 4ms idle, these policies chose to wait. By the time the 4ms timer ended, the system became busy again, successfully avoiding an incorrect sleep transition. This proves that the timeout mechanism effectively filters out short idle pulses

### 5.5.2 Ramp

Under the Ramp workload, the Simple Predictive Policy exhibited the highest energy consumption (1.2019 J). This is because the policy relies solely on historical data without a timeout safety net; when the workload transitioned from a short (6ms) to a long (100s) idle period, it caused the system to incorrectly remain in the high-power IDLE state for the entire 100s, leading to massive static power waste. In contrast, the Fixed Hybrid Policy demonstrated significantly lower energy consumption (0.3200 J). This is because it incorporates a 4ms timeout mechanism that forces the system into SLEEP when a prediction fails, thus saving energy during the majority of the long idle period; however, it still suffered from false positive sleep transitions during the 6ms noise bursts ($6ms > 4ms$), causing unnecessary wake-up penalties. To further mitigate these penalties and compensate for the rigidity of the fixed parameter, the Adaptive Hybrid Policy introduced a **dynamic timeout mechanism** that learns the "noise floor" from history. This allowed the system to automatically adjust the safety timeout to exceed 6ms, effectively filtering out the noise bursts while preserving deep sleep capabilities, resulting in the lowest overall energy consumption (0.2836 J).