**Slide 1:** Good morning esteemed members of the jury, professors. My name is Tareq Md Rabiul Hossain Chy, and I had the privilege to undertake my Master's 1 internship at Telecom SudParis. Under the insightful guidance of my supervisors, Sophie Chabridon from Télécom SudParis, as well as Denisse Munante from ENSIIE, I have focused my research on 'ENACT - ENergy efficiency through ArChitectural Tactics for Software Engineering.'

**Slide 2:** During my presentation, I'll start by discussing the key motivation behind this internship. Next, I'll outline the primary goals we set out to achieve through our research during the internship. I'll then share the specific contributions I've made to the research. Finally, I'll summarize the findings and discuss the next steps for future work in this area.

**Slide 3:** In the modern world, the devices we use every day our phones, computers, and smart home gadgets are major consumers of electricity. Shockingly, over 63% of the world's electricity is generated using fossil fuels, which are also the leading cause of the greenhouse gas emissions responsible for 75% of global climate change. The more electricity we consume, the more we contribute to this global crisis. So, making our software more energy-efficient isn't just good for our electricity bills—it's essential for the health of our planet.

**Slide 4:** The main objective was Exploring tactics for software energy efficiency improvement. For Research Question 1 (RQ1), we explored tactics for improving energy efficiency. In RQ2, we focused on automating the integration of tactics to lower energy consumption. The approach involved using genetic improvement. Three sub-questions were identified: RQ2.1 aimed to verify if improvements in response time and memory consumption (individually or together) could result in energy reduction. Experiments were conducted using the GIN tool, with optimized versions compared to their original versions using JoulerJX to assess energy impact. In RQ2.2 we investigated the feasibility of integrating code refactoring into Genetic Improvement (GI), identifying necessary extensions to the GIN tool. Finally, RQ2.3 explored the extent to which the integration of code refactoring could genetically enhance software to reduce energy consumption, testing its impact on energy efficiency.

**Slide 5:** The primary focus of this research work is to explore tactics for enhancing software energy efficiency. Several tactics can be used to achieve this goal, including Design Patterns, and Code Refactoring, Architectural Tactics. **Design Patterns:** Design Patterns are typical solutions to commonly occurring problems in software design. **Code Refactoring:** This is the process of restructuring existing code without changing its external behaviour (i.e. functionalities) to improve its readability, maintainability and reduce the energy consumption of the software. Based on code refactoring qualities we select code refactoring.

**Slide 6:** We conducted a comprehensive literature review in which we attempted to extract data concerning which refactoring techniques have an impact on reducing energy consumption. We found that Convert Local Variable to Field to till Encapsulate Field consistently showed positive results in reducing energy consumption across various scenarios. We selected those refactoring to integrate into the genetic improvement tool. Conversely, Extract Local Variable, Extract Method, Inline Method, and Introduce Indirection exhibited varying effects on energy consumption, depending on the specific context.

**Slide 7:** Genetic algorithms are commonly used to mimic natural evolution to find optimal solutions to complex problems. Genetic Improvement (GI) is a specialized type of genetic algorithm used to improve software. In GI, an initial set of software solutions is initialized in the initialization phase and evaluated based on a "fitness" measure, which assesses how well the software performs in the fitness function evaluation phase. If the solutions don't meet predefined criteria, they are altered or "mutated" to introduce variation in the mutation phase. This new set is then re-evaluated. The process repeats until an optimal solution is found, based on the stopping criteria. One research paper examined the usability of Genetic Improvement (GI) tools. Only 2 tools, Gin and PyGGI, could be readily applied for software improvement. Gin is specifically designed for the Java ecosystem. Selected for research due to Java’s lower energy consumption compared to Python.

**Slide 8:** Gin is a toolbox designed for Genetic Improvement (GI) experiments, focusing on Java software. It helps in two main ways: **1. Pre-processing**: Gin identifies key methods, or 'hot methods,' in software that are most likely to benefit from optimization. It does this by measuring and ranking each method's execution time or any other fitness function. **2. Search Space Analysis**: Gin explores possible code edits that can improve the software. It tests these edits by running a test suite on the modified code, recording details like whether the edit works, how long it takes, and if it passes all tests.

**Slide 9:** All of these mean values have a confident error margin, i.e., the relative standard deviation or the coefficient of variance was lower than 5%. **Notice that when a multi**-criteria fitness function is used to optimize a code, each fitness value is optimized in a lower percentage than optimizing the code by using a single fitness function. For instance, while the percentage of the reduced execution time for Tactic 1 and Tactic 3 is equivalent, the percentage for the consumed memory was penalized from 21.43% to 13.33%. Instead of this penalization, Tactic 3 is the most effective tactic to reduce energy consumption for the Triangle program. It could be due to the energy consumed by programs does not only depend on execution time but also depends on the use of CPU. Therefore, if we optimize the memory consumption it could help to reduce the use of CPU, thus reducing energy consumption.

**Slide 10:** We can also observe this reduction of energy consumption in Figures (a) and (b) which show the histogram and the box-plot of the data collected from 30 executions.

**Slide 11:**

**Slide 12:** We can also observe this reduction in energy consumption in Figures (a) and (b) which show the histogram and the box plot of the data collected from 30 executions by applying Tactic 3.

**Slide 13:**

**Slide 14:** We can also observe this reduction in energy consumption in Figures (a) and (b) which show the histogram and the box plot of the data collected from 30 executions by applying Tactic 1.

**Slide 15:** Based on the class diagram, Gin's Edit class serves as the foundation for different types of code edits, with LineEdit and StatementEdit as its subclasses. We've chosen StatementEdit as the right class for adding code refactoring techniques because it specializes in modifying code statements, which is closely related to code refactoring. This integration aims to create more energy-efficient versions of the program.

**Slide 16:** In my research, I have presented tactics for creating energy-efficient software, with a particular focus on code refactoring. I have also discussed JoularJX, a tool designed for monitoring the energy consumption of software. Additionally, I introduced the Gin tool, which integrates selected code refactoring techniques. My contributions to research are as follows: **1.** I identified code refactoring as an effective tactic for energy efficiency, thus answering Research Question 1 (RQ1). **2.** I automated the JoularJX tool for energy consumption monitoring using Bash scripts. **3.** By optimizing programs with Gin, I achieved up to a 51.53% reduction in energy consumption, thereby addressing Research Question 2.1 (RQ2.1). **4.** I identified opportunities for integrating refactoring techniques within Gin’s StatementEdit class, answering Research Question 2.2 (RQ2.2).

**Slide 17:** We will work on enhancing the Gin tool by integrating selected code refactoring techniques, but first, we'll embed a code smell detection feature. We'll use a search-based approach, modifying Gin's source code to spot code smells efficiently. After this, we'll integrate refactoring techniques like Convert Local Variable to Field and others. Ultimately, we aim to test whether these improvements lead to energy-saving software, addressing Research Question RQ2.3.

**As** we can observe in Table, the Gin toolbox was able to optimize the Triangle program by using Tactic 1 - minimizing the execution time, Tactic 2 - minimizing the memory consumption, . The energy consumption was reduced by 13.9% by the optimised program (see Column 3 of Rows 1, 2) by applying Tactic 1The energy consumption was reduced by 16.43% by the optimised program (see Column 5 of Rows 1, 2) by applying Tactic 2. **Finally**, the Gin toolbox was able to optimize the Triangle program by using Tactic 3 - minimizing execution time and memory consumption. The energy consumption was reduced by 28.11% by the optimized program (see Column 7 of Rows 1, 2) by applying Tactic 3. **We observe** that for the Triangle program, the best tactic for reducing energy consumption was Tactic 3 - minimizing execution time and memory consumption. This tactic was able to reduce by 28.11% the energy consumed by the Triangle program (see Column 8 of Rows 1, 2).

**As** we can observe in Table, the Gin toolbox was able to optimised the Greatest Common Divisor(GCD) program by using Tactic 1 - minimizing the execution time. The energy consumption was reduced by 51.16% by the optimised program (see Column 3 of Rows 3, 4) by applying Tactic 1. **Moreover, t**he Gin toolbox was able to optimize the Greatest Common Divisor(GCD) program by using Tactic 2 - minimizing memory consumption. The energy consumption was reduced by 51.26% by the optimized program (see Column 5 of Rows 3, 4) by applying Tactic 2. **Finally,** the Gin toolbox was able to optimize the Greatest Common Divisor(GCD) program by using Tactic 3 - minimizing execution time and memory consumption. The energy consumption was reduced by 51.53% by the optimized program (see Column 7 of Rows 3, 4) by applying Tactic 3. **We observe** that for the Greatest Common Divisor(GCD) program, the best tactic for reducing energy consumption was Tactic 3 - minimizing execution time and memory consumption. This tactic was able to reduce by 51.53% the energy consumed by the Greatest Common Divisor(GCD) program (see Column 8 of Rows 3, 4).

**As we** can observe in Table 5.1, the Gin toolbox was able to optimize the Rectangle program by using Tactic 1 - minimizing the execution time. The energy consumption was reduced by 20.78% by the optimized program (see Column 3 of Rows 5, 6) by applying Tactic 1**. Moreover,** the Gin toolbox was able to optimize the Rectangle program by using Tactic 2 minimizing the memory consumption. The energy consumption was reduced by 17.6% by the optimized program (see Column 5 of Rows 5, 6) by applying Tactic 2. **Finally, t**he Gin toolbox was able to optimised the Rectangle program by using Tactic 3 - minimizing execution time and memory consumption. The energy consumption was reduced by 14.45% by the optimised program (see Column 7 of Rows 5, 6) by applying Tactic 3. **We observe** that for the Rectangle program, the best tactic for reducing energy consumption was Tactic 1 - minimizing the execution time. This tactic was able to reduce in 20.78% the energy consumed by the Triangle program (see Column 8 of Rows 5, 6).