Article:
Ajraou, H., Ward, R., Farbiz, F. et al. (2026) ACT-FLEX: A symbolic and generative AI integration architecture for generalisable and explainable robotic disassembly tasks. Robotics and Computer-Integrated Manufacturing, 101. 103277. ISSN: 0736-5845

https://doi.org/10.1016/j.rcim.2026.103277

Highlights

ACT-FLEX: A Symbolic and Generative AI Integration Architecture for Generalisable and Explainable Robotic Disassembly Tasks

Houdefya Ajraou, Rob Ward, Farzam Farbiz, Erich Graf, Yuan Miaolong, Yan Shijun, John Oyekan

- Integrates ACT-R cognitive architecture with LLMs for robotic disassembly tasks.
- Enables generalisable and explainable robot behaviour across diverse product morphologies.
- Uses CAD diagrams and VLMs to generate disassembly instructions.
- Achieves high success rates in simulation and physical robot experiments.
- Supports sustainable circular manufacturing through intelligent, adaptable disassembly pipelines.

ACT-FLEX: A Symbolic and Generative AI Integration Architecture for Generalisable and Explainable Robotic Disassembly Tasks

Houdeyfa Ajraou[a], Rob Ward[a], Farzam Farbiz[b], Erich Graf[c], Yuan Miaolong[d], Yan Shijun[d], John Oyekan[e]

[a]*School of Electrical and Electronic Engineering, University of Sheffield, Mappin Street, Sheffield, S1 3JD, United Kingdom*
[b]*Institute of High Performance Computing, A*STAR, 1 Fusionopolis Way, Singapore, 138632, Singapore*
[c]*University of Southampton, University Road, Southampton, SO17 1BJ, United Kingdom.*
[d]*Advanced Remanufacturing and Technology Centre (ARTC), A*STAR, 3 Cleantech Loop, Singapore, 637143, Singapore*
[e]*Department of Computer Science, University of York, Heslington, York, YO10 5DD, United Kingdom*

Abstract

The increasing demand for personalised products poses new challenges for sustainable manufacturing, particularly for the large-scale disassembly required for circular economy initiatives. Traditional robotic systems lack the flexibility and generalisability needed to adapt across diverse product types and robot morphologies. In this work, we present ACT-FLEX, a cognitive architecture inspired by psychological models of human cognition. We used ACT-FLEX to show that pre-trained Large Language Models (LLMs) as well as Vision-Language Models (VLMs) can enable flexible, explainable and generalisable robotic disassembly across robot morphologies and tasks. Towards this, our architecture incorporates both symbolic reasoning,

Email addresses: hajraou1@sheffield.ac.uk (Houdeyfa Ajraou), r.a.ward@sheffield.ac.uk (Rob Ward), Farzam_Farbiz@ihpc.a-star.edu.sg (Farzam Farbiz), E.W.Graf@soton.ac.uk (Erich Graf), Yuan_Miaolong@artc.a-star.edu.sg (Yuan Miaolong), Yan_Shijun@artc.a-star.edu.sg (Yan Shijun), john.oyekan@york.ac.uk (John Oyekan)

multimodal perception and morphology-aware action generation to support decision-making in dynamic environments. We validate ACT-FLEX across simulated and physical experiments involving multiple disassembly scenarios, product configurations and various robotic platforms including UR10, KUKA iiwa, Panda Franka and uArm Swift Pro. Results show transferability across robot platforms and robustness to product variations, with success rates of up to 80% in physical trials. This work demonstrates a step toward realising transparent, retaskable and sustainable robotic systems for Industry 5.0.

Keywords: ACT-R, Cognitive Architecture, Generative Pre-Trained Transformers, Robotics, Manufacturing

---

1. Introduction

Due to the increasing customisation and personalisation of products, recycling these products in order achieve circular economies will present a new set of critical challenges to sustainability and the goals of circular economy. This is because industrialising the disassembly of varying products and making it economically viable for Small Medium sized Enterprises (SMEs) or companies in the supply chain would require a new way of programming industrial automation in order to account for the large variety of products and their conditions.

For example, products such as laptops, computers, televisions including old Cathode Ray Tube (CRT) displays, mobile phones, printers and automotive batteries [1] have a wide range of varieties and product families addressing different market sectors. The mobile phone sector has up to 50 product families under which there are a number of product varieties. In the automotive sector, Electric Vehicle batteries are another source of growing product diversity. Each automotive company typically offers 3-7 product families with each product family having 5-12 variants defined by vehicle class, drive-train, charging standards and platform [2]. These products contain useful and valuable materials such as gold, silver, neodymium and others that are very much within their useful life range and should be recovered. However, the use of humans to perform the undesirable, dirty and dangerous nature of recycling would require large labour wages thereby making the process less commercially viable. As a result, in order to support industrialised re-manufacturing and future sustainability endeavours, flexibility in automation becomes an important consideration.

The ability of humans to adapt rapidly and be flexible towards disassembly of various products is as a result of their brain structure. The brain provides humans with the capability to process information and adapt rapidly to changes in situations. This has inspired investigations into how the brain is structured and how robotic agents can be endowed with similar capabilities. As a result, various cognitive architectures have been developed with the motivation, design and structure of these architectures based on various methods. These cognitive architectures aim to represent how information is processed from sensory inputs all the way to decision making in various contexts. While some architectures focus on the connections between the biological processes of the brain, others aim to emulate more human-like behaviour without a biological connection [3]. Nevertheless, these architectures provide an explainable structure and framework within which to anchor the development of generalised cognitive agents for various contexts while achieving Human-like artificial intelligence (HLAI).

Up until now, most of the modules such as the semantic and procedural memories in cognitive architectures have relied on symbolic processing that involve hand-coding of features and rules. These make them fit and effective for a particular known use case for which they have been designed for [4]. Nevertheless, the performance of a tuned architecture falls drastically when it is translated to another use case. Furthermore, handcrafting reasoning and decision making, though possible in cognitive architectures, is time-consuming and would not capture all the nuances of various use cases. This leads to cognitive "rigidity".

On the other hand, Large Language Models (LLMs) provide the cognitive "flexibility" required due to their training on large scale knowledge. This large scale knowledge and the ability to use prompts to obtain information regarding a large variety of use cases provide the flexibility required for a cognitive architecture to move from one use case to another without being retuned or retrained. Nevertheless, there is no cognitive psychology standard or framework on how to harness LLMs to develop flexible cognitive agents in manufacturing. This leads to adhoc and siloed developments that are often specific to a particular manufacturing use case. This means that LLMs are not being used to their full potential.

Furthermore, research has shown that LLMs suffer from black box reasoning that rely on their on static internal representations. These internal representations are not sufficiently grounded in the sensory inputs derived from the external world as the use cases change [5]. This limits their ability

to reason reactively or to update their knowledge leading to hallucinations and errors in reasoning. As a result, LLMs currently fall short in replicating human cognition appropriately leading to the exhibiting of human-level capabilities that are unhuman-like [6, 7].

In the disassembly use cases mentioned above, these unhuman-like and sometimes unrealistic reasoning results (hallucinations) lead to LLM derived plans that do not take the morphology of the robot agent being used into consideration. This leads to unrealistic control plans. This is because, while powerful, LLMs are limited because their knowledge is based on "ideal" or "textbook" data [8]. As a result, they fail when confronted with the reality of uncertainty in product condition and variability across end of life products that are caused by varying wear, damage, deformations, missing parts or as in our case, variation between product families of units and robot morphologies. From the above, there is a need to flexibly update the LLM's knowledge when needed without expensive and extensive retraining cycles.

This need and the lack of flexibility in LLMs as well as in cognitive architectures particularly across various product families and robot morphologies is the key contribution and research gap that this paper addresses. Specifically, our contributions are as follows:

1. We set out the expected characteristics of a flexible architecture and the psychology informed requirements it must have to address the above re-manufacturing challenges.
2. We used the psychology informed requirements to develop and validate a psychology inspired cognitive architecture which combines symbolic cognitive frameworks with powerful pre-trained neural models to enable flexible, transparent and trustworthy robotic systems.
3. We show that by using off-the-shelf pre-trained models and without further training, our cognitive architecture is transferable and works across various disassembly tasks as well as robot morphologies thereby demonstrating its potential for generalisability in circular re-manufacturing.
4. We develop and introduce a new cognitive architecture called **A**daptive **C**ontrol of **T**hought and Rational with **FLEX**ibility across Robot tasks and morphologies (**ACT-FLEX**) that uses a number of Language Models to achieve a large scale implementation of neural mechanisms across various cognitive buffers. This is unlike previous implementations of ACT-R where researchers have implemented neural mechanisms for one or two buffers. This serves as a step towards a neurolog-

ical plausible computational brain.

In section 2, we discuss the state of the art attempts to use LLMs and Cognitive Architectures in disassembly tasks. In section 3, we present psychology informed requirements that a cognitive architecture must achieve if it is to be flexible and generalisable across various tasks. We also present our flexible cognitive architecture together with the quantitative metrics we used for evaluating it. Experimental results are presented in section 4 with discussion and limitations in section 5. We conclude in section 6 with a summary as well as future work.

2. Related Work

As discussed above, in order to achieve to sustainable and viable circular manufacturing, there is an increasing need for automation that is capable of flexibly and rapidly adapting across varying product families and robot morphologies towards the disassembly of products. Robotic disassembly of electric batteries is receiving a lot of attention due to the transition to cleaner sources of energy. A recent example is the work conducted in [9] where an unscrewing process for further disassembly was investigated. Towards this, they implemented a Convolutional Neural Network YOLO (You Only Look Once) nano architecture to detect four screw heads for unscrewing by five screwdriver bits. Similarly in [10], Zhou et al made use of an innovative Particle Swarm Optimisation (PSO)-Pareto algorithm to schedule collaboration between a human and robot for screw removal during the disassembly of end-of-life (EoL) products. The use of optimisation algorithms in human-robot collaborative disassembly was also investigated by [11] in which they implemented a Non-dominated Sorting Genetic Algorithm II (NSGA-II) to find the optimal way to disassemble a waste power battery modules WPBMs from the Tesla Model S. In [12], the main contribution was a task planner for the disassembly of the Audi A3 Sportback e-tron's Lithium batteries. Their task planner used YOLO for object detection and then decided what component should be removed. The task planner then proceeds with screws removal and loops until the goal is completed. A similar approach was used in [13] except that they augmented the computer vision algorithms with used data access from the manufacturer to obtain the product assembly structure and combination.

Other techniques have also used methods such as mixed-integer linear programming (MILP) [14], Q based reinforcement learning [15], multi-objective

policy optimisation [16], swarm-based algorithms [17] as well as knowledge graphs [18]. However, these methods are mostly focused towards one product and one robot morphology. Furthermore, they are static (not flexible) due to their restricted knowledge base or cost functions that often need to be manually redesigned or updated tediously to achieve optimal results when tasks or robot morphologies change.

Though technologies such as Language Models offer a wider knowledge base, they are also hampered by other factors that impact their flexibility. In order to solve this challenge, various researchers have investigated combining them with other algorithms as we now discuss below.

2.1. Applying and combining LLMs with key existing algorithms

The structure of LLMs provide them with the capability to store vast amounts of knowledge and the opportunity for humans to use that knowledge on demand through the use of natural language prompts. Though powerful, LLMs have been known to hallucinate and provide false information which is highly dangerous in safety critical applications and has reduced trust in them.

This challenge is further compounded by the unpredictable conditions of End Of Life products as well as their variety which makes it difficult for fully automated systems to handle disassembly tasks [19]. As a result, an automated system must have the cognitive flexibility to deal with such scenarios. So far, this flexiblity has been handled by humans. As a result, majority of work in the application of LLMs in disassembly have focused on a human-robot collaborative process. For example, in [20], a human–robot collaborative disassembly (HRCD) sequence planning model involved developing a constraint graph constructed from semantic documents of the targeted EoL product. The constraint graph was combined with a Dirichlet Bayesian Network (DiBN) to generate feasible sequences based on the constraint graph. The output of the DiBN was then passed to a fine-tuned LLM in order to quantitatively analyse the output and augment it with other knowledge about the EoL product.

Similarly, the use of LLMs with Graphical Neural Networks (GNNs) have been trialed. In [21], GNNs were used to encode the semantic relationships between components in the object to be disassembled while LLMs were used to contextualise information. In [20], a Reinforcement Learning module was used to generate feasible sequences for the product disassembly which were then validated by the LLM. However, the use of GNNs and other graph like

approaches raise performance flexibility issues due to their heavy reliance on the quality and accuracy of the input graph structure. This is particularly difficult in scenarios where the capability of the robot to move from task to task rapidly and flexibly is required.

Another approach in the literature utilises a combination of Evolutionary Algorithms and LLMs. In [22], Huang et al. used Evolutionary Algorithms to reproduce a self-reflective strategy that humans use. LLMs were used to generate heuristic rules based on the job to be completed and paired with the statistical properties of the dataset. This enabled the system to use information about the tasks as well as scheduling constraints when generating the rules. In this scenario, these rules led to constraints on the Evolutionary Algorithm population which in turn optimise the rules generated by the LLMs. While not used for disassembly, the approach could be adapted for use on a disassembly task. Nevertheless, this requires that the hyperparameters of the Evolutionary Algorithm are well tuned to enable effective search and generation of disassembly sequence. This becomes more complicated when hybrid product manufacturing lines are under consideration.

In dealing with the arrangement of disassembly tasks in hybrid disassembly lines of multiple products, a mixed-integrated programming model has been used in a Dueling Deep Q-Network (Duel-DQN) combination with a reinforcement learning enabled LLM [23]. Guo et al.'s framework enabled numerous manufacturing line constraints to be taken into consideration within the set of generated solutions [23]. The LLM within the framework converted human text into disassembly sequences that were then used to configure the manufacturing line and train the Reinforcement Learning Agents in the changed environment. This then produced control algorithms appropriate to the task at hand. In this setup, carefully produced prompts are needed to generate the right sequences. Furthermore, the prompt response needed to be converted into a format that was compatible with the reinforcement learning algorithm.

In all the above solutions, the hallucinations that LLMs could potentially produce introduce safety concerns that need to be addressed. In order to enhance the reliability, accuracy and transparency of LLMs, Knowledge Graphs are often integrated with them. Knowledge Graphs give a structured representation of the domain and enable traditional knowledge expressions to be embedded into an intelligent system. This addresses the black-box nature of LLMs providing a check and balance mechanism to hallucinations while making them more transparent [24]. However, building a knowledge

Graph of a domain is often very time consuming and requires an extensive knowledge of the domain in order to be effective. Furthermore most of the applications of LLMs have been adhoc with no underlying structure that can be used to generalise across tasks.

2.2. Cognitive psychology inspired architectures and LLMs

Cognitive psychology inspired computational architectures (also known as Cognitive Architectures) have been studied since the 1970s [4]. This has resulted in a range of unique architectures that have been proposed for generalist and specialist contexts. Two of the most commonly used arcitectures are ACT-R (Adaptive Control of Thought—Rational) and SOAR (State, Operator, And Result) architectures [25, 26].

Both SOAR and ACT-R have been designed to model human cognition and simulate the way humans think, reason, solve problems and learn. Despite sharing this goal, they differ significantly in design philosophy, structure and use cases. For example, SOAR was heavily influenced by the development of Artificial Intelligence (AI) Algorithms and was modelled as a general problem solving architecture focused on general intelligence, human like reasoning and decision making. On the other hand, ACT-R is more grounded in human cognitive psychology as it was developed to replicate and explain data from empirical studies in psychology and neuroscience.

The original architectures have been augmented by researchers to better mimic human cognition; for example by using reinforcement learning to represent the modules in the architectures (SOAR-9). This enabled its general architecture to make decisions for unseen scenarios which its symbolic rules do not cover [25]. Furthermore, the ACT-R architecture has been extended to robotics as ACT-R/E [27], an embodied cognitive architecture for human–robot interaction. ACT-R/E allows a robot to embody a theory of human cognition, anticipating human errors or limitations (e.g., forgetting or distraction) and adapting accordingly. This improved human–robot collaboration by making robots more predictive and adaptive. ACT-R has also been applied to interactive storytelling robots, ensuring coherent responses and explainable reasoning [28].

In [29], Lieto et. al. introduced a goal-directed and dynamic knowledge generation mechanism into the architecture. This enabled an agent to automatically re-frame or re-formulate the available knowledge in the declarative memory for a particular task. With the development of the transformer architecture, this has been replaced by LLMs [30].

In [31], Siyu et al. introduced LLM into the ACT-R architecture to support design for manufacturing decision making tasks. Their aim was to use LLMs to guide the replication of perception, memory, goal-setting and action in the ACT-R architecture. They fine-tuned LLAMA2 [32] with their dataset composed of simulated examples of trials with prompt requests. In a series of experiments and the resultant comparisons, LLM-ACTR outperformed LLAMA2 and reduced hallucinatory events. Nevertheless, the limitations included the need to fine-tune LLAMA2 thereby making the study relatively inflexible and difficult to transfer to tasks outside manufacturing.

Researchers have since experimented with more complex hybrid cognitive architectures such as C3I1 (Three Cognitions, One Intelligence) that was introduced by Wray et al in [33]. C3I1 is made up of three layers of cognition inspired by human cognition biology: Proto-cognition, Micro-cognition, and Macro-cognition. In their architecture, proto-cognition's role was to decrease the scale and complexity of the raw input data [34]. To achieve this, the researchers experimented with two swarming-based mechanisms: SODAS, a decentralized hierarchical clustering shown in [35] and marker-based stigmergy, a topological sorting algorithm. The Micro-cognition component was used for small tasks that were familiar and was based on the ACT-R framework [36] for problem-solving. In addition, it used Bayesian Learning mechanisms for retrieving relevant information from the declarative memory as well as Reinforcement Learning to tune procedural actions through the use of successes and failures. The Macro-cognition was used to represent broader task problem solving. In this case, they made use of the SOAR architecture.

Though these cognitive architectures have developed a set of tools, there is limited application of them in robotic disassembly. Notably, [37] used SOAR to improve a robot's ability to remove screws from laptops. In their approach, the semantic memory stored information about laptop models, screw positions and previously checked holes. This enabled the robot to identify orientation of the laptop as well as recall prior actions that have been conducted on the laptop. This reduced trial time by over 60% and demonstrated the value of cognitive memory for flexible and knowledge-intensive disassembly.

2.3. Research Gap and Summary

Though both the SOAR and ACT-R architectures serve as theoretically grounded frameworks to qualitatively and quantitatively develop as well as program intelligent systems for a variety of tasks, in our estimation there has

been no work on their use in developing autonomous systems for disassembly across different robot morphologies and use cases. Furthermore, a current limitation of these architectures is their reliance on manual and hard-coded symbolic knowledge, which motivates research of combining them with modern learning models such as LLMs.

Several methods of combining Cognitive Architectures and Large Language Model agents have been proposed, yet there is little work in regard to applying them in physical robotic environments. Additionally, while LLMs have been investigated and combined with other computational tools, all the approaches have been adhoc, with little theoretical grounding and bespoke to a particular application. This restricts their transferability to another application.

In this work, we address these limitations and provide a cohesive cognitive psychology grounded framework that can be used to task disassembly robots across multiple domains and across various robot morphologies. We fuse together the benefits of the structure of the existing ACT-R cognitive architecture with state-of-the-art LLMs to provide the flexibility missing in previous efforts.

3. Methodology

In this section, we discuss the requirements we adhered to in the development of our cognitive architecture. We then follow with a design and implementation of the cognitive architecture as well as the quantitative metrics that were used to evaluate it.

3.1. Cognitive Psychology informed Architectural Requirements

The overall purpose of a cognitive architecture is to make it possible for an agent to apply all of its available knowledge towards reasoning and deciding on which actions to apply within the context of a task. In order to quantitatively and qualitatively test if an agent has achieved this, Laird et al. defined a set of requirements that a cognitive architecture must meet [25, 38, 39]. These requirements have been objectively defined through various experiments and analysis that have been carried out in literature over the decades. We summarise these requirements (R) as well as our corresponding proposals (P) below (We also refer the reader to Table 1 for a concise summary of the requirements and our proposals):

R0: The cognitive architecture must be fixed for all tasks: This means that the architecture can not change from one task to another. As a result, such an architecture provides a stable platform for acquiring and using knowledge over various timescales from the immediate up to the entire lifetime of the robotic agent.
Proposal 0: Based on the ACT-R architecture, we propose the structure in Figure 1. This structure did not change from task to task during our experiments on different use cases and robot morphologies.

R1: Realise and make use of a symbol system: The use of a symbol system to represent objects makes it possible to attach actions and operations to symbols and then recombine the symbols flexibly in novel ways to solve a problem. It also enables a robotic agent to operate on the symbols "internally" and find optimal ways of working with that symbol before actually working on it in the physical external world.
Proposal 1: In order to achieve this requirement, we made use of the digital equivalent of physical objects in the form of Computer Aided Diagram (CAD) diagrams. The CAD diagrams were comprised of various components in digital format that made up a product. These digital components were used as a system of symbols towards which we could attach actions and operations. The use of CAD diagrams also gave us an opportunity to test our automation strategies in a simulator before transferring to the real world. Nevertheless, our current pipeline does not explicitly have internal simulation in the loop during control.

R2: Represent and effectively use modality-specific knowledge: In order to obtain symbols of physical objects in the environment, sensing modalities and computational structures are required to detect them and translate them into formats that can be used by a robotic agent. Depending on the complexity, novelty or diversity of the task, the computational resources of the agent must be sufficient in order to achieve real time processing and control especially in a dynamic environment. Furthermore, depending on the novelty and complexity of the task, detecting and reasoning about spatial properties of the object such as distance, rotation and inversion is necessary.
Proposal 2: We used a real time camera and Vision based Language Model (VLM) to detect objects, their positions, geometry as well as their spatial properties in the workspace. Furthermore we made use of a vision-based LLM to translate CAD diagrams into potential instructions to use further up in the control pipeline. All the agent's processes fitted into a laptop run-

ning Ubuntu 22.04 with 32GB RAM while the Vision Language Model for detection and CAD-translation agents were offloaded to DashScope via API calls.

R3: Represent as well as effectively and efficiently access large bodies of diverse knowledge: A robotic agent must have access and use a large body of knowledge derived from the nature of the tasks that the agent will be doing, complexity of the environment, robot's morphology regularities and dynamics as well as diverse knowledge of other agents in the environment such as humans and tools. Furthermore, this knowledge must be diverse, including memories of experiences, facts of the world, skills and knowledge of about other agents.

Proposal 3: To achieve this requirement, we made used of a chain of off-the-shelf pre-trained Large Large Models (LLMs) that have been trained on diverse amount of information across various industrial sectors and contain facts about the world, tools and robot regularities. In our architecture, we also made use of a LLM that converts step by step instructions into robot instructions specific to its morphology.

R4: Represent and effectively use knowledge with different levels of generality: The robot must be able to make use of the current conditions in the environment as well as the current state of the task, in making decisions. This should occur across various tasks that the agent is tasked with.

Proposal 4: We make use of continuous sensing of the environment to detect where the components are and what steps need to be taken next. As long as the target component is not moved after detecting its location, the pipeline will process it accurately. For example, if a component moves while the robot processes another component, the pipeline will still detect the next component and process it.

R5: Represent and effectively use diverse amounts of knowledge: The robotic agent must be able to update its knowledge seamlessly especially in situations where the tasks are novel and its knowledge is limited.

Proposal 5: We made use of a knowledge Library within which users could add wiki-hows and other documents relevant to the task being done. Accessing this knowledge library is done intelligently through the use of information extracted by the LLMs at the beginning of the pipeline. Adding this module ③ in Figure 1, enables users to transparently access what knowledge the robot is using to complete its tasks while at the same time giving them the ability to add documents and content useful for the robot to carry out its

tasks.

R6: Represent and effectively use beliefs independent of perception: The agent must have the ability to represent and reason about hypothetical situations which is a necessary component in planning. In order to achieve this, a symbol system is required for manipulating the symbols towards creating a plan both for novel situations, partial known situations and already known situations. This capability enables the agent to make a decision based on not just its current situation but also on its memory of previous situations as well as predictions of future situations. This capability is often supported by the ability to conduct internal mental simulations.
Proposal 6: Our current system lacks an internal mental simulation. Nevertheless, access to the broad knowledge via LLMs enables our agent to harness this when considering novel situations.

R7: Represent and effectively use rich, hierarchical control knowledge: The robotic agent must have a rich representation for control including having to deal with a rapidly changing environment, across different timescales. This might mean that knowledge about actions are organised and conducted hierarchically as well as decomposing some of its planned actions into sequences of simpler actions.
Proposal 7: We used a hierarchical architecture that reads sensor inputs and decomposes the detected task's state into a sequence of steps and corresponding actions to achieve the goal of the task specified by visually annotated CADs. We make use of a TAMP system that converts high-level coarse commands into fine motor commands for use on the robot end-effectors.

R8: Incorporate innate utilities: In view of the agent's continuous operation over its life time, there are states in the environment, the agent's morphology or in the task itself that could impede the agent's ability to carry out its task successfully.
Proposal 8: Towards supporting the agent, we introduce the ability to use LLM prompt templates to define robot morphologies being used in various tasks.

R9: Initiate, represent and manage goals at multiple time scales: Different tasks have different goals and also have different time scale requirements for completion. Goals can be complex as well as diverse and this means that a robotic agent must actively manage varying goals or interrupt them as required. For example, it might become important to interrupt a task due to a safety concern.
Proposal 9: Collaborative robots have been programmed to operate at low

speeds for human safety and to stop whenever a human interferes with their motion. In this work, we rely on this underlying safety mechanism to stop the collaborative robots even when in the middle of a task. However, we recognise that requirement **R9** is more nuanced than that.

R10: Access, represent and effectively use meta-cognitive knowledge This requirement involves embedding knowledge about the robotic agent into itself as well as knowledge about its own knowledge (meta-knowledge). This becomes important especially in situations of dealing with novel tasks. However, the exact range of the necessary meta-cognitive knowledge is not clear and complete meta-cognitive knowledge is not required. In humans, they make use of existing knowledge to find out knowledge gaps when presented with novel tasks and then use this information to explore for more knowledge to plug the gaps.

Proposal 10: For this requirement, we made use of a if-else function to let the human know when the robot does not understand how to proceed on a task. We also rely on the human in order to embedded or provide the right knowledge to the robot agent during tasks in module ③. We also use module ⑨ and ⑩ in Figure 1 to embed the knowledge of the current robot morphology into our architecture.

R11: Support a spectrum of bounded and unbounded deliberation: The complexity of tasks has an impact on the computational resources required for planning for the tasks. Furthermore, the tasks impose time constraints that need to be respected if the they are to be completed successfully by the agent. As a result, the robotic agent must have sufficient computational resources in order to do this. At one end of the computational demand spectrum is for the robot to achieve planning from first principles while at the other end of the specturm is the robot being reactive in response to changes in the environment and task completion states. Between this two extremes, the agent must use its knowledge to balance between a reactive and deliberative response. In the first case, a reactive behaviour is possible provided if the agent's knowledge of its environment and the tasks are completed. In the case where the knowledge is not complete, more deliberation and planning from first principles is required.

Proposal 11: Due to the ability to inject knowledge into our architecture via module ③, we can give the robotic agent as much knowledge as required regarding a task and the other agents it might be interacting with. Furthermore, modules ⑤ and ⑥ enables us to test and check if the robot has all it needs to complete its tasks.

R12: Support diverse, comprehensive learning: The robotic agent must have the capability to learn over its lifetime and recall the appropriate learnt skill to solve a task. Although general learning mechanisms exist, they are often biased to the specific task under consideration as well as the robot peculiarities. In order for a cognitive architecture to be useful, it needs to have the capability to learn all different types of tasks, the knowledge related to them and not be restricted by the learning mechanism used.
Proposal 12: The modules ④, ⑧ and ⑨ gave us the ability to access the inherent knowledge contained within the LLMs through the use of prompts.

R13: Support incremental, online learning: The agent must have the capability to modify its knowledge based on its interaction with its environment. The knowledge of a current situation should be capable of being stored for future reuse. Furthermore, the ability to store and retrieve the appropriate knowledge or experience must be capable of real time evaluation even as more and more knowledge is added over time.
Proposal 13: Our current architecture does not have the capability to achieve this requirement by itself. More specifically, it relies heavily on humans to give it up-to-date knowledge. Nevertheless, it has the capability to retrieve knowledge in real time but not store it.

For the convenience of the reader, we summarise the above requirements and our proposals in Table 1 below. We also showed if our architecture met the requirements presented above or not.

Architectural Requirement	Proposal Summary	Met?
R0: Fixed for all tasks	Architecture structure remains constant across tasks and robot morphologies.	Yes
R1: Use a symbol system	CAD diagrams serve as symbolic representations for attaching actions and simulating tasks.	Yes
R2: Use modality-specific knowledge	Vision-based LMs and CAD-to-instruction translations detect and represent objects.	Yes
R3: Access large, diverse knowledge	LLMs trained on diverse domains plus morphology-specific instruction translation.	Yes
R4: Use different levels of generality	Continuous sensing tracks state and task progress.	Yes
R5: Use diverse amounts of knowledge	Knowledge library allows users to add task-relevant documents accessed via LLMs.	Yes
R6: Use beliefs independent of perception	Lacks internal simulation but leverages LLM knowledge for novel cases.	Partial
R7: Use hierarchical control knowledge	Hierarchical TAMP system decomposes high-level goals into fine motor commands.	Yes
R8: Incorporate innate utilities	Achieved in our design via robot morphology templates.	Yes
R9: Manage goals at multiple time scales	Relies on cobot safety mechanisms to interrupt tasks when needed.	Partial
R10: Use meta-cognitive knowledge	If-else checks alert humans when robot lacks knowledge.	Partial
R11: Support bounded/unbounded deliberation	Knowledge injection and module checks allow balancing reactive and deliberative responses.	Yes
R12: Support diverse learning	LLM prompts access inherent knowledge across multiple modules.	Yes
R13: Support incremental, online learning	Can retrieve but not store new knowledge without human intervention.	No

Table 1: Requirements R0–R13 with proposal summaries and evaluation of whether met (color-coded).

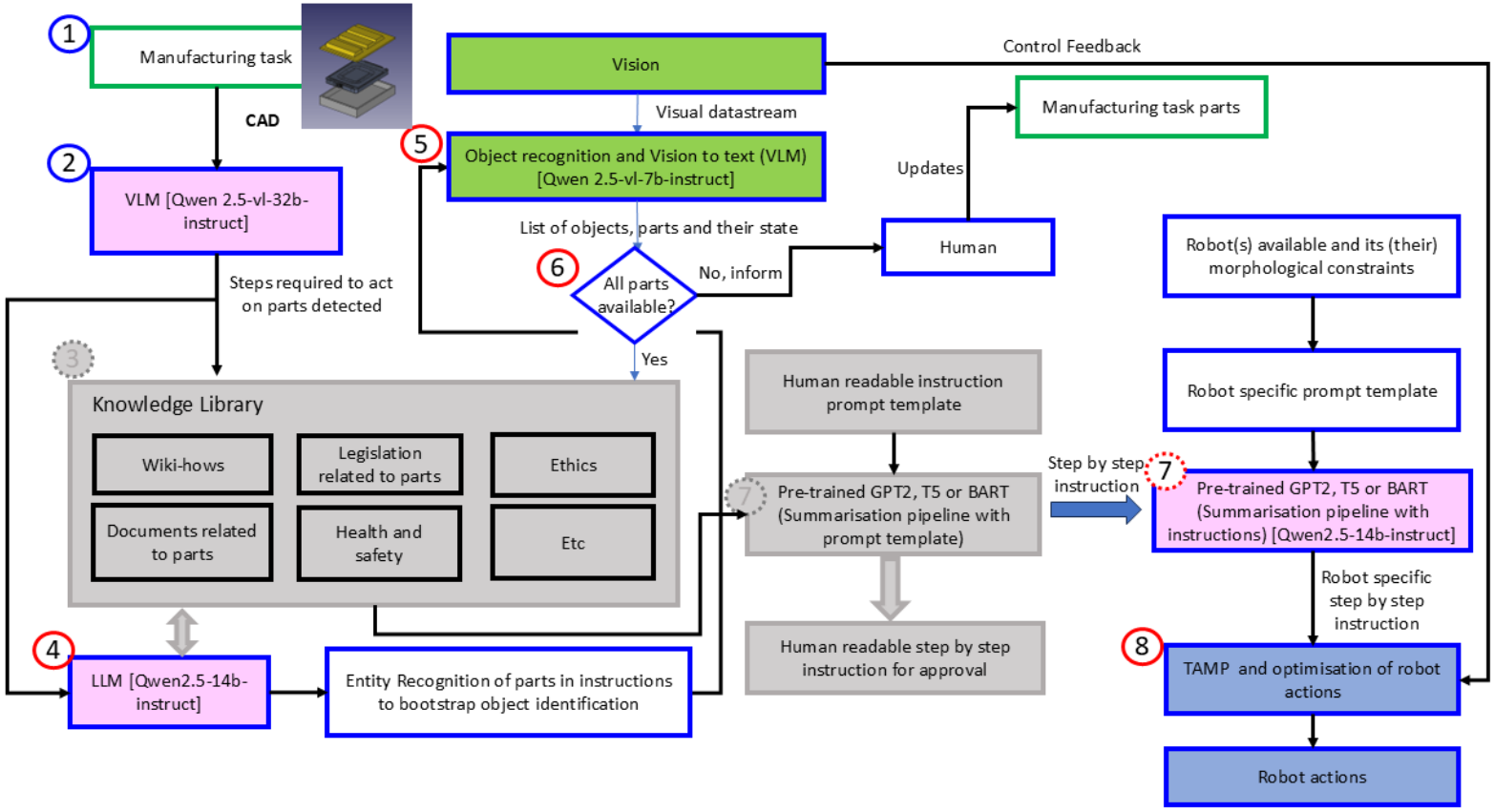

Figure 1: Implementation of our proposed architecture called Adaptive Control of Thought and Rational with FLEXibility (ACT-FLEX) across Robot tasks and morphologies. We show how various Language Models were used the architecture. In the current implementation, we did not use the greyed out regions while the red circles indicate computational loops that are repeated until the task is finished. The blue circles are active at the beginning when a robot starts a task.

3.2. The choice of ACT-R as a bootstrap cognitive architecture

In literature, cognitive architectures are broadly categorised into symbolic, emergent and hybrid [4]. Symbolic based cognitive architectures rely mostly on the manipulation of symbols during information processing. As a result, objects in the environment need to be converted into symbols after sensing for further manipulation. Furthermore, symbolic based architectures make use of explicit rules and logical inferences (if-then rules and production systems) to process these symbols into actions. Furthermore, knowledge is represented declaratively ("as facts") and procedurally ("how to use those facts"). As a result, they are good for human interpretability and explainability. The disadvantage with them is that they are rigid, brittle and very difficult to adapt rapidly and flexibly to new use cases.

On the other hand, emergent based cognitive architectures are often based on computational tools like neural networks. In this situation, the knowledge of a task is stored in the strengths of connections between neurons and not in symbolic rules. Such systems are often more biologically plausible, robust to noise and able to transfer flexibly across tasks due to their learning ability. Nevertheless, it is hard to interpret their internal representations due to their lack of reliance on symbols and human interpretable rules.

Hybrid architectures such as CLARION [40], ACT-R and SOAR combine the best of both worlds mitigating the weaknesses of both approaches. Furthermore, compared to the rest of the hybrid architectures, these three architectures embedded the most plausible human cognitive processes of sensory, working, semantic, procedural, episodic memories. Choosing one of these would enable us to combine the explainability and interpretability of symbolic architectures with the flexibility of emergent based architectures. Of the hybrid architectures, ACT-R and SOAR have stood the test of time and seen the largest academic and industrial support with over 40 years of development [4].

In comparing ACT-R and SOAR, SOAR implements knowledge reasoning through the execution of context based rules (the symbolic part) while emergent neural computational tools like reinforcement learning are used to integrate continuous learning into the intelligent agent. Furthermore, SOAR's general computational paradigm is based on providing a blueprint or framework that resembles the cognitive capabilities exhibited by a human [41]. In other words, it provides a framework that replicates the "processing modules" in a human cognitive process but not necessarily the connections and

timings between them. As a result, SOAR's cognitive neural grounding and psychological fidelity is more abstract when compared with ACT-R [42].

On the other hand, ACT-R was designed to model the reaction times and error patterns found across human memory, attention, learning and decision-making. ACT-R, was constructed around separate specialised cognitive modules that correspond to a psychological or brain subsystem. Its philosophy is that cognition arises from the interactions between these specialised modules (buffers). The interaction between these buffers is governed by production rules (symbolic). As a result, the use of production rules enables us to control the flow of information between the buffers and hence embed explainability. In our work, we used neural LLMs (emergent) to implement the various buffers. Using computational neural approaches for the buffers in this large-scale way moves us towards a fully computational artificial brain. This is unlike previous extensions to the ACT-R architecture [43, 7] that predominately make use of symbols in their buffer implementations or have more localised neural implementations.

In cognition, different brain networks are specialised for different functions. This is reflected in the ACT-R architecture in which it is used to model human neurological framework from the brainstem all the way to the higher level of decision making processes. As a result, these areas have different neural substrates and functionalities meaning that their technical implementation would also need to be different. In our implementation, we have also used different Language Models for the buffers and modules in our architecture. For example, Vision Language Models (VLMs) are used for vision based processing similar to the visual cortex in the brain while Large Language Models are used for processing textual data, decision making in constructing robot actions similar to the frontal cortex in the brain. These actions are passed to a Task And Motion Planner (TAMP) module for low level motor control. This approach makes our architecture closer to biologically plausibility compared to previous approaches in [43, 7]. We explain our design and implementation further in the next section.

3.3. Design and implementation of the proposed Cognitive Architecture

We propose the cognitive architecture shown in Figure 1 which we call **A**daptive **C**ontrol of **T**hought and Rational with **FLEX**ibility across Robot tasks and morphologies (**ACT-FLEX**) based on our previous work in [44]. This architecture contains majority of the components of the Adaptive Control of Thought and Rational (ACT-R) shown in Figure 2 as well as LLMs

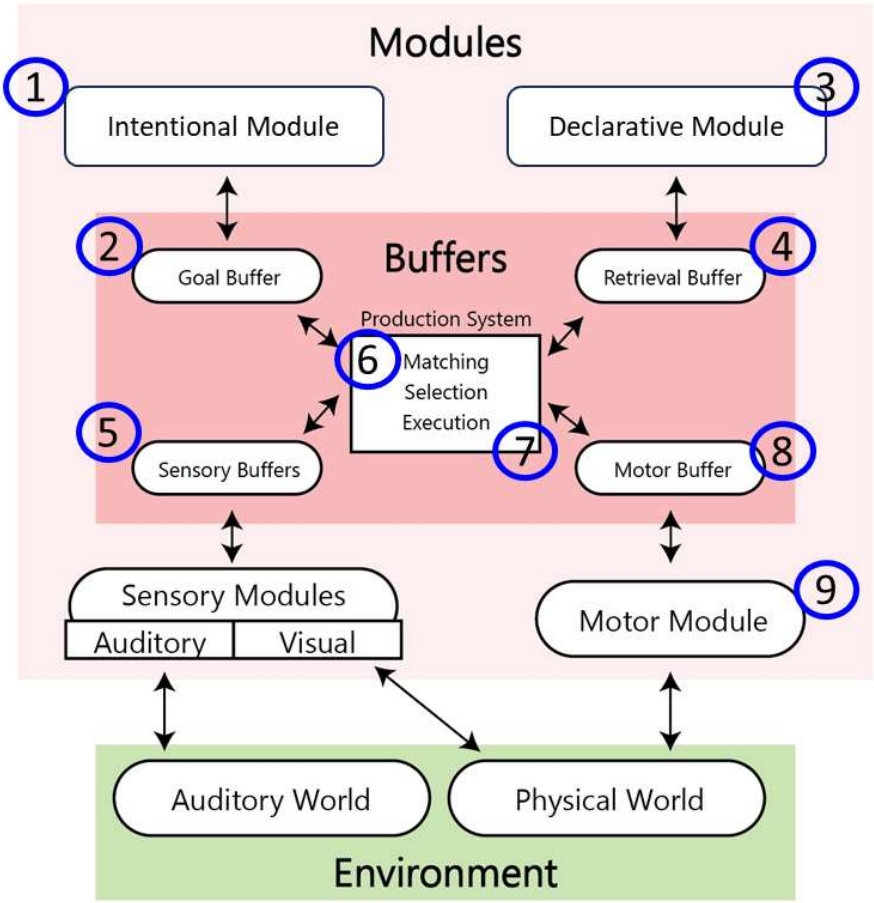

Figure 2: The ACT-R architecture and its relationship to our proposed ACT-FLEX architecture.

to enable rapid generalisation, flexibility and transferability. We now discuss how the ACT-FLEX architecture was designed for implementation using the current ACT-R as a guide.

Both architectures start with the human having a task to complete. This is communicated to both architectures via the use of the intentional module in ACT-R and the manufacturing task module in ACT-FLEX (Modules ① in both Figures 2 and 1. The reader is advised to read both Figures in parallel). In ACT-FLEX, the goal or task is provided in the form of exploded CAD diagrams which are decoded and stored in the equivalent of the Goal Buffer in the ACT-R architecture (Modules ②). In ACT-FLEX, we used a VLM to

decode the CAD diagram into a series of n discrete steps in natural language text. These steps contained the components to be acted upon as well as the action to be performed on the component. We then acted on each step until the total number of steps were completed.

The modules ③ in both architectures enable users to add in any knowledge in the form of documents that is relevant to the task being completed. In ACT-FLEX, this could be regulations, ethics or wiki-hows which can then be read as well as parsed by a Large Language Model. In ACT-R, this knowledge is often encoded in the form of **if-else rules** which makes it very cumbersome to use by novices. Retrieving this knowledge in ACT-FLEX could take the form of Natural Language Processing constructs, Language Large Models or Knowledge extraction tools such as Knowledge Graphs and Ontologies.

In this paper, we used a LLM in module ④ of Figure 1 to process the steps from the VLM in module ②. For each step, the relevant components were extracted via a LLM that does entity recognition ④. For example, in a step with the instruction "lift the yellow lid", the relevant component would be "yellow lid". In the future, we foresee that regulations or constraints relating to a component in a step could be added to the knowledge library for extraction and usage by the LLM in ④.

Module ⑤, in both architectures are the sensing modules. In our work, we make use of camera sensors to detect objects in the workspace. Using the entity component extracted in ④, we use camera data streams and a vision based Large Language Model (VLM) to find its 2-dimensional pixel coordinates via key-point or bounding-box detection in ⑤. Furthermore, we use the VLM to convert the detected component into internal symbols for further architectural manipulation. The 2-dimensional pixel is converted into a 3-dimensional location using a depth camera and an ARuco marker whose offset to the robot base is known. Based on the 3D coordinates of the components, another LLM in Module ⑦ is used to generate the start position and the target position for action by the robot. Since this is a disassembly task, we assume that the target position is predefined. In other words, the components are disassembled from a product and placed in a pre-specified location for later transport.

In module ⑥, we check if all the parts or tools (Symbols) needed for the manufacturing task are present in the workspace. This is similar to the matching module in the ACT-R architecture. If there are missing parts, we alert the human to provide the additional parts or tools so that the robotic

agent can continue its tasks. In ACT-FLEX, by combining the knowledge from the knowledge module ③ (Declarative module in ACT-R) and the output of the VLM in module ② (Goal Buffer module in ACT-R), we use a LLM in module ⑦ (Execution module in ACT-R) to construct a list of robot actions (Selection module in ACT-R) for execution. In ACT-FLEX, we provide a human readable template that can be used to retrieve the robot actions so that human oversight and transparency is maintained. Currently, our implementation has the options between two actions: pick-and-place and push-drag. Furthermore, module ⑦ in ACT-FLEX enables us to inform our architecture about the robot's unique capabilities and its morphology.

We achieve this by enabling users to define a robot specific template to reconfigure our architectures particularly the trajectories required for the robot's end effectors. The trajectories of the robot are then passed to the Task And Motion Planning modules for planning and converting into motor specific motions to achieve the required trajectories ⑨. These motions are then realised on the robot.

Once an action is chosen, the robot arm performs it with the target location data as parameters to Module ⑧. The above process is repeated until the sequence of steps provided by the VLM in Module ② are completed. The Pseudocode for ACT-FLEX is shown in Algorithm 1 and 2 while the associated prompts for the Language Models are shown in Appendix A.

3.4. Quantitative Metric for Pipeline Evaluation

In order to measure the performance of our pipeline system, we implemented a quantitative metric that draws inspiration from the reward function introduced by [45]. In their research, they made use of a "inserting" (into a component) reward function for their Reinforcement Learning pipeline. The reward function was formulated such that it consisted of these five terms (see Equation 1 in which λ_i and α_i were weighting terms):

- Euclidean distance between the goal position and the actual position of the components ($\|x_o - x_g\|_2$).
- Angular difference ($\arcsin\big(\text{clamp}(\|d_a\|_2, 0, 1)\big)$.
- Distance from each fingertip to the object ($\min\big(e_0 - \sum_{i=0}^{4} f_i,\, 0\big)$).
- Action vector (e.g. joint velocities) a.

Algorithm 1 ACT-FLEX Pseudocode: Our main procedure is the DISASSEMBLY TASK which takes as input a textual description of the ***task*** to be carried out; real time ***Image*** of the workspace as captured by a RGB-D camera and the ***CAD Diagram*** of the product to be disassembled together with the visual instructions (See Figure 4).

procedure DISASSEMBLY TASK($Task$, $Image$, $CADDiagram$, $DepthImage$, $CameraIntrinsics$)
    $steps \leftarrow$ GETSTEPSVLM($CADDiagram$)
    for all $step \in steps$ **do**
        (componentName, componentColor) $\leftarrow$ EXTRACTRELEVANTOBJECTLLM($step$)
        $Pos2D \leftarrow$ FIND2DPOSITIONKEYPOINT($componentName$, $componentColor$, $Image$)
        $Pos3D \leftarrow$ 2D_3D_PROJECT($Pos2D$, $Image$, $DepthImage$, $CameraIntrinsics$)
        $action \leftarrow$ GENERATEACTIONLLM($Pos3D$, $step$)
        EXECUTE($action$)
    end for
end procedure

- Torques applied τ.

$$
\begin{aligned}
r = \lambda_1 \exp\Big[- \big(\alpha_0 \|x_o - x_g\|_2 + \alpha_1 \cdot 2 \arcsin\big(\text{clamp}(\|d_a\|_2, 0, 1)\big)\big)\Big] \\
+ \lambda_2 \min\big(e_0 - \sum_{i=0}^{4} f_i,\, 0\big) + \lambda_3 \|\mathbf{a}\|_2^2 + \lambda_4 \|\tau\|_2^2
\end{aligned} \tag{1}
$$

Since we were performing disassembly instead of assembly, we only focus on the Euclidean distance between the goal position and the actual position of the components. We did not have any specifications on the angle, the motion optimisation or other precision metrics for manipulating the robot's tool. We had only one target goal for the disassembled components, which is that all the components should be dropped and stacked on each other. We ignored the height dimension (z) and reduced our evaluation metric function to Equation 2 where parameters are explained in Table 2:

Algorithm 2 Find2DPositionKeyPoint: We used a Gradient Search algorithm to move a key point towards the centre of a component's surface and return the updated two dimension key point. Due to the use of Qwen language model, we had to use TranslateZHLLM to translate the component color into Mandarin.

procedure FIND2DPOSITIONKEYPOINT(componentName, componentColor, Image)
    componentNameZH ← TRANSLATEZHLLM(componentName)
    BBOX ← DETECTBOUNDINGBOXVLM(componentNameZH, Image)
    croppedImageScene ← CROPIMAGEFROMBBOX(Image, BBOX)
    Pos2D ← DETECTKEYPOINTVLM(componentNameZH, croppedImageScene)
    ColorCodeHex ← GETCOLORCODEHEXLLM(componentColor)
    ColorHSV ← CONVERTHEX2HSV(ColorCodeHex)
    NewPos2D ← GRADIENTSEARCH(ColorHSV, Image)
    return NewPos2D
end procedure

$$r = \lambda \exp(-\alpha_0 \|x_o - x_g\|_2) \tag{2}$$

Symbol	Definition	Units
r	Reward assigned to a single component	–
λ	Weighting coefficient for positional accuracy	–
α_0	Scaling factor for distance penalty	–
$x_0 = (x, y)$	Current component Position vector (2D)	m^2 components
$x_g = (x, y)$	Goal component Position vector (2D)	m^2 components

Table 2: Definitions of the symbols in our reward function.

This formula allows for a score value between 0 and 1. If the position is what is expected, the difference between the current position and the goal position will be equal to 0 which will make the exponential function equal to 1. If however, there was some distance between the current and the goal position, the exponential function will be less than 1.

Figure 3: A battery pack assembly

4. Experiments and results

In this section, we discuss our assumptions, experimental setup as well as results from simulation and physical experiments.

4.1. Assumptions

In our experiments, we assumed that: **(i)** The product to be disassembled is made up of sub-components that are modular. This assumption covers Electric Vehicle (EV) battery packs that often contain battery cells as shown in Figure 3. These packs vary in design depending on the manufacturer [46] and we replicated this as a use case as shown in Figure 4; **(ii)** There are no mechanical components holding the modules together. This is true for electronic products like mobile phones and electric vehicles batteries where adhesives are used for structural bonding and sealing. This ensures efficient thermal and electrical management as well as the protection of contact points and encapsulation of electronics.

4.2. Evaluating VLM for CAD based instructions

Since a Vision Language Model (VLM) was crucial in providing human explainable instructions to our architecture, we discuss and analyse this part of our architecture first. Towards this, we assessed the performance and reliability of several VLMs in converting visual instructions into textual step by step instructions.

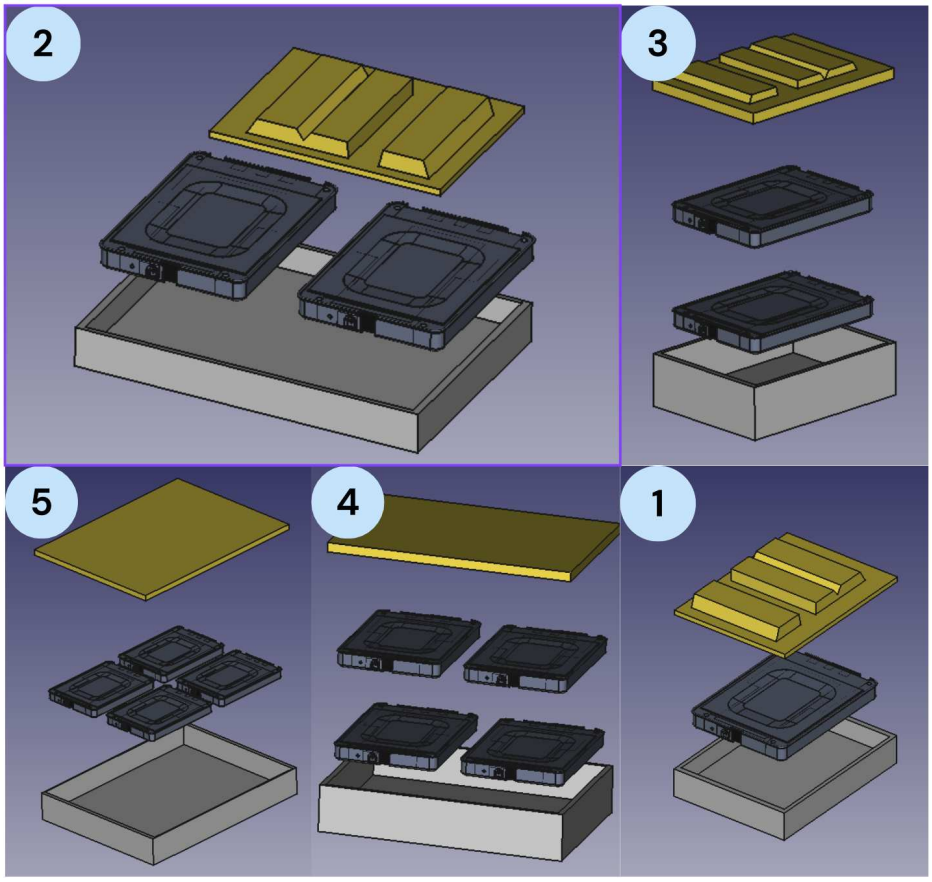

Figure 4: The five different EV battery cases used in the simulation experiment set up

A dataset of 9 images was curated as shown in Figure 5. Each image represented an exploded view of an assembled structure across various use cases. Each image was queried to each VLM independently with a specific text prompt that asks the VLM to detect all the visible components in the visual instruction image along with their color. To ensure consistency in the output, we used a structured textual output that consisted of a list of objects. Each object in the list had two attributes: name and color.

For each image-model pair, we manually review the predicted objects and applied a scoring rubric applied as follows:

- Maximum score per image-model pair is two times the real total number of components in the image.

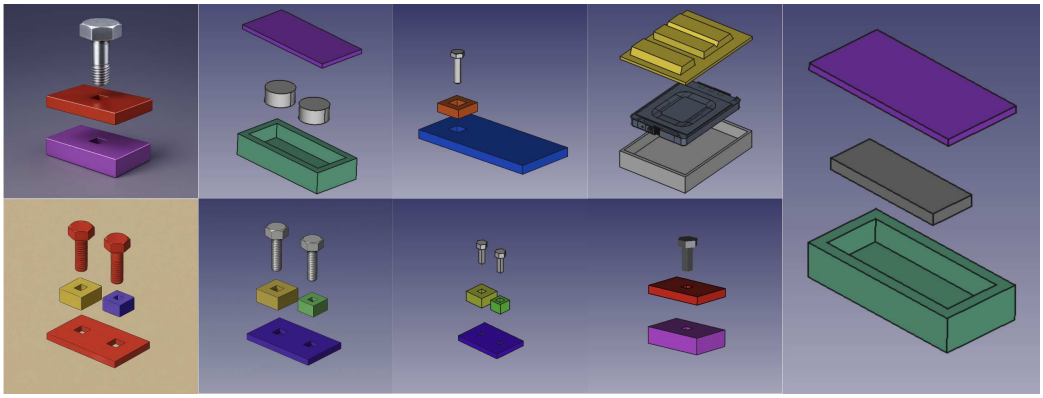

Figure 5: The curated image dataset that was passed to the VLMs to test their ability to extract the required steps to complete a task.

- For each hallucination (e.g. an extra part predicted), a penalty of 0.5 point is applied.
- For each missing component in the predicted list, a penalty of 0.25 points is applied.
- For each component that is correctly predicted, we assign two points, the maximum number of points a component can have.
- For each component that is correctly predicted, but with the wrong color, one point is given, half of its full credit.

The evaluation was conducted in a consistent visual setting which involved side-by-side comparisons across models to ensure fair judgment.

This approach aimed to expose the performance of the VLMs when it came to listing the various parts in an exploded view diagram for supporting the generation of steps to perform a disassembly later in our pipeline. We evaluated 10 models in total namely: Mistral-Small3.1 [47]; Qwen2.5-VL-32B-Instruct [48]; Qwen2.5-VL-7B [48]; Gemma3-12B [49]; Gemma3-27B [49]; Gemma3-4B [49]; LLama3.2-Vision [50]; Qwen2.5-VL-32B [48]; LLava [51] and Minicpm-V [52].

Table 3 shows more information about the models tested, including size and context windows. All these models were used in the Ollama package, apart from the Qwen2.5-VL-32B-Instruct, which was interfaced with with the DashScope-Alibaba Cloud API.

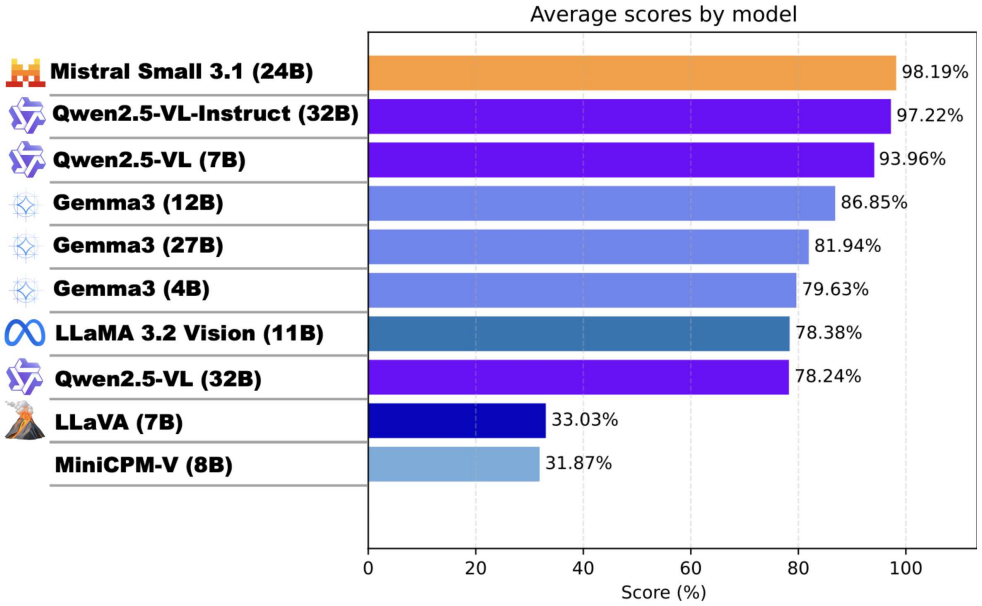

Figure 6: The Figure shows the VLMs that were tested together with their scores. The colors indicate the brand of the models used and do not have any relationship with the results or dates the models were released, its size or any other attributes.

VLM Name	Size in billion parameters (B)	Size in Gigabytes (GB)	Context Window	Release Date
Small Mistral 3.1	24	15	128K	17 Mar 2025
Qwen2.5 VL 32b Instruct	32	?	125K	28 Jan 2025
Qwen2.5 VL (7b)	7	6.0	125K	28 Jan 2025
Gemma3 (12b)	12	8.1	128K	21 Mar 2025
Gemma3 (27b)	27	17	128K	21 Mar 2025
Gemma3 (4b)	4	3.3	128K	21 Mar 2025
LLaMA3.2 Vision	11	7.8	128K	25 Sep 2024
Qwen2.5 VL (32b)	32	21	125K	28 Jan 2025
LLaVA	7	4.7	32K	5 Oct 2023
MiniCPM-V	8	5.5	32K	6 Aug 2024

Table 3: Information of the VLMs used in our experiments

While Small-Mistral3.1 obtained the best score in our qualitative system, we used Qwen2.5-VL-32B-Instruct model in our pipeline. This is because

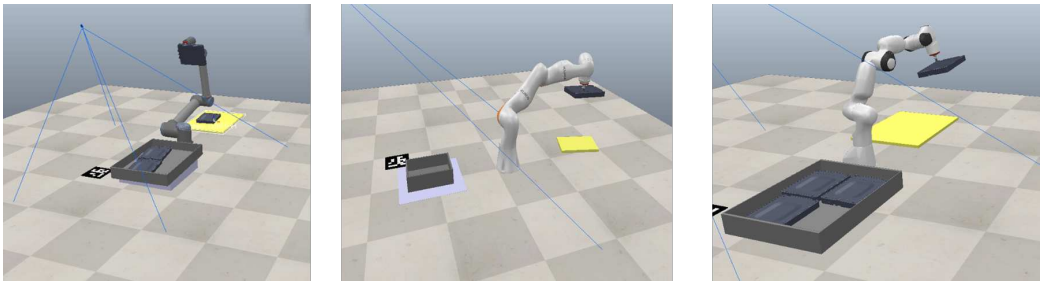

Figure 7: Showing the simulation setup on various robot morphologies and use cases. The slight changes in the components show that our robot is able to deal with variations in position.

Small-Mistral3.1 was released in March 2025 after Qwen2.5-VL-32B-Instruct and we had already started experimenting with Qwen2.5 VL. Furthermore, there was a minimal difference of less than 1% in the qualitative scores of both models.

4.3. Simulation Experiments

As shown in Figure 4, we designed 5 configurations of EV batteries in CAD design. We enumerated each of them and called them Single Battery (**SB**), Double Battery (**DB**), Single Battery Double Stack (**SBDS**), Double Battery Double Stack (**DBDS**) and Four Battery (**FB**) respectively. In simulation experiments, we ran 18 tests for each of the 3 robot morphologies. In each run, the batteries were placed in different 3D positions. This was to test the transferability of our architecture onto various robot morphologies and to meet requirement **R0**.

We tested our architecture on three robot arms (UR10, KUKA iiwa, Panda Franka) tasked with disassembling the five EV battery configurations (See Figure 7). For each robot and battery configuration pair, we ran 18 tests with variations in the location of the battery pack. As a result, this translated into 18 x 3 x 5 = 270 simulation experiments with corresponding results. We conducted the simulated experiments in CoppeliaSim configured with the Bullet 2.8 engine. The robot arm movements and inverse kinematics were configured and controlled with OMPL (Open Motion Planning Library) algorithm obtained from MoveIt2 and ROS2 (Robot Operating System 2)[53].

Due to the difference in morphology, each robot needed a different set of offset parameters as shown in Table 4. The first two dimensional offsets X and Y are mainly dependent on the ArUco (Augmented Reality marker)

marker position relative to the robot base link. The Z depends on the end effector position on the robot as well as the length of the vacuum cup used in our experiments.

Parameter	Definition	Units	Panda	KUKA iiwa	UR10
$Offset_X$	X axis offset between the ArUco marker and the robot base in meters	m	-0.4	-0.4	-0.4
$Offset_Y$	Y axis offset between the ArUco marker and the robot base in meters	m	-0.8	-0.8	-0.9
$Offset_Z$	Z axis (Height) offset in relation to the end effector during picking	m	0.05	0.05	0.02

Table 4: Offset values for the various robot morphologies used simulation.

4.4. Simulation Results

As mentioned above, we evaluated our proposed pipeline on 18 x 5 x 3 experimental runs, that is 18 trials for each pair robot battery pair. The λ and α_0 parameters in Equation 2 were set to 1. In our experiments, we used Equation 2 to analyse the robot's performance using three metrics. The three metrics investigated the performance of the robot using: (i) the battery cells position after the disassembly; (ii) The battery cells' and battery cases' position after the disassembly and the (iii) the positions of all the components, that is the battery cell, battery lid and battery case after the disassembly process. We did this across all the battery configurations shown in Figure 4. During each experiment, the battery was placed in a different position in order to test the robustness of our architecture to variations in object positions. Table 5 shows a snapshot of the variations in the initial positions of the battery cases when testing the KUKA iiwa robot arm on the Double Battery (DB) configuration.

Tables 7 to 8 show the average score obtained using the three metrics mentioned above. In the tables, it can be seen that the KUKA iiwa robot morphology out performed the other two robots used. One possible reason for this relates to the UR10's morphology which has a different articulation profile to the KUKA iiwa. On the Panda arm, the end effector does not move as much downwards as the KUKA iiwa. These contributed to the differences in the results obtained.

Trial #	INIT x (m)	INIT y (m)	INIT z (m)	Lid dist. (m)	Base dist. (m)	Cell–1 dist. (m)	Cell–2 dist. (m)
1	0.0	0.0	0.0	0.622	0.000	0.638	0.576
2	0.0	0.0	0.1	0.585	0.000	0.638	0.590
3	0.0	0.0	0.2	0.586	0.210	0.492	0.705
4	-0.1	0.0	0.0	0.622	0.000	0.664	0.586
5	-0.1	0.0	0.1	0.585	0.001	0.666	0.516
6	-0.1	0.0	0.2	0.650	0.000	0.665	0.624
7	0.1	0.0	0.0	0.617	0.000	0.626	0.596
8	0.1	0.0	0.1	0.593	1.285	1.078	0.603
9	0.1	0.0	0.2	0.620	0.000	0.565	0.661
10	0.0	-0.1	0.0	0.596	0.000	0.621	0.595

Table 5: Example per-trial component Euclidean distances for the KUKA iiwa robot on the Double Battery (DB) configuration. The distances are computed in 2 dimensional with respect to the target position ($x = 0.0, y = 0.7$), while the base component is evaluated relative to its initial position. Ten representative trials are shown for illustration. ***INIT (x, y, z)*** denotes the initial X, Y and Z position in meters we initiate the battery structure relatively to the software's origin (0, 0, 0)

	SB	**DB**	**SBDS**	**DBDS**	**FB**
PANDA	0.458984	0.428999	0.688767	0.462649	0.365794
UR10	0.508434	0.413590	0.685940	0.522849	0.284800
KUKA iiwa	0.509499	0.646511	**0.789004**	0.719501	0.283926

Table 6: Evaluation of robot arms performance based on battery cells' positions only after disassembly.

	SB	**DB**	**SBDS**	**DBDS**	**FB**
PANDA	0.726807	0.617827	0.729853	0.536994	0.469809
UR10	0.664274	0.542475	0.676069	0.542835	0.397702
KUKA iiwa	0.712803	0.733653	0.736377	**0.744951**	0.399595

Table 7: Evaluation of the robot arms performance based on battery cells and cases positions after disassembly.

	SB	**DB**	**SBDS**	**DBDS**	**FB**
PANDA	0.687665	0.648450	0.757091	0.578305	0.489217
UR10	0.646038	0.545744	0.686271	0.571829	0.409317
KUKA iiwa	0.653525	**0.764870**	0.756128	0.755045	0.415716

Table 8: Evaluation of robot arm performance based on every component positions after disassembly.

4.5. Physical experiments

Similar to the simulations, we used ArUco markers in the physical experiments as the robot arm moves in Cartesian 3D space to perform pick and place actions. In the physical experiments, we used two robots namely the uArm Swift Pro and the xArm. Due to its small footprint, we used the uArm Swift Pro for disassembling a toy example of small scale electronic device namely a Raspberry Pi while we used the xArm robot to disassemble a larger scale representation of an Electric Vehicle battery. We now discuss both experiments below.

4.5.1. Applying the uArm Swift Pro to disassembly a small scale electronic device

Figure 8 shows the visual instruction provided to our architecture with a Raspberry Pi and a battery represented as a silver block. We used (x,y,z) = (0.134, -0.085, -0.018) meters as the positional offsets between the ArUco marker and the robot base. In Figure 9, the detected key points for different components can be seen with the right Figure showing the key point (green dot) for the battery cell and the left Figure showing the key point for the lid.

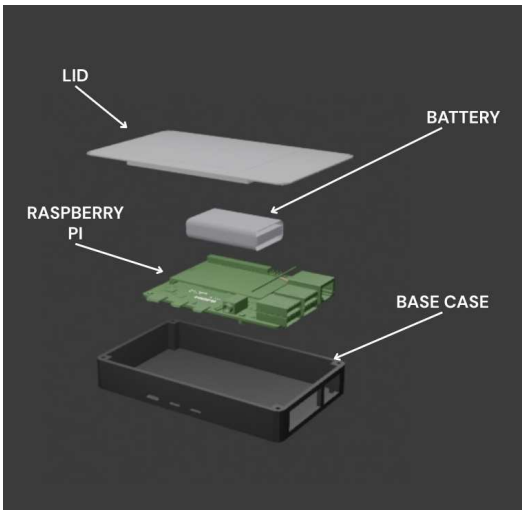

Figure 8: Showing an exploded view of a small electronic device representation, consisting of a Raspberry Pi circuit board, battery, lib and black casing.

In our experiments, we explicitly prompted the VLM not to interact with the Raspberry Pi and the case's base in the disassembly prompt by using:

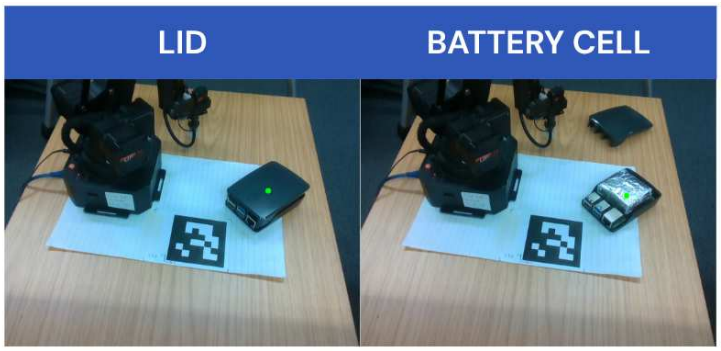

Figure 9: Showing the uArm and the detected key points at two different stages with the left Figure showing detection of the lib and the right Figure the detection of the battery cell.

```
....
DO NOT INCLUDE ANY STEPS THAT INVOLVES TOUCHING THE BLACK BASE
DO NOT INCLUDE ANY STEPS THAT INVOLVES TOUCHING THE GREEN CIRCUIT
....
```

This is because it was difficult to get the suction cup to lift the green circuit board and subsequently the black case. We ran 20 trials with variations in the position and the orientation of the Raspberry Pi and achieved a success rate of 65%. Table 9 shows the details of the 20 runs as well as the reason if a failure occurred.

4.5.2. Apply the xArm robot to disassemble a EV battery representation

In applying the xArm for disassembly, we used a single battery (labelled as **SB**) configuration. Figure 10 shows the disassembled battery on the left and the visual instructions provided to the architecture on the right. Figure 11 shows the detection of the lid on the left as well as the detection of the battery cell on the right. We used (x, y, z) = (0.22, 0.2, 0.11) meters as the position offset for the xArm. We ran 10 trials using various positions and orientations of the battery representation. We obtained a success rate of 80% with the only errors caused by the failure of the suction cup to grasp the battery cell as shown in Table 10.

Trial #	Success?	Reason of Failure
1	True	NA
2	True	NA
3	True	NA
4	True	NA
5	True	NA
6	False	Failed to read data through the camera
7	True	NA
8	True	NA
9	False	Failed to suck up the battery cell
10	False	Failed to suck up the battery cell
11	True	NA
12	False	Failed to locate the battery cell
13	True	NA
14	False	Failed to suck up the lid
15	True	NA
16	False	Confused the lid and the robot
17	True	NA
18	True	NA
19	True	NA
20	False	Bad moves, pushed the battery leaf instead of keeping the end effector up like usual

Table 9: Performance evaluation of the uArm Swift Pro using boolean validity as a measure of success. The reasons for failure are also represented in the column on the right.

5. Discussion

In this paper, we present a cognitive architecture called ACT-FLEX that is capable of transferability between tasks as well as robot morphologies. Our cognitive architecture was inspired by cognitive psychology particularly in the way neural substrates in the human brain processes from sensory inputs to decision making mechanisms and then onto action.

We demonstrate that flexibility in robotics can be achieved by taking inspiration from these informational processing pathways. Towards this, ACT-

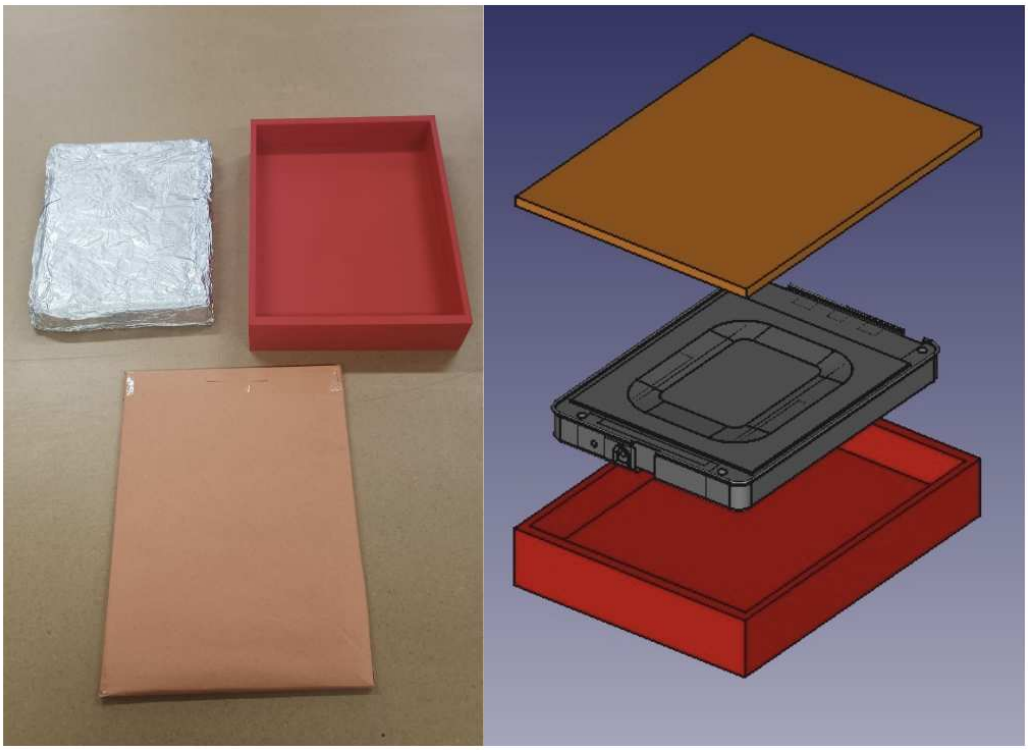

Figure 10: Physical representation of an Electric Vehicle disassembled battery case on the right with visual instructions presented as an exploded view on the left.

FLEX provides a computational structure of this informational processing pathway. Using ACT-FLEX as an anchor for Language Models, we were able to achieve flexibility and explainability through a combination of emergent (VLMs/LLMs) and symbolic (cognitive structure) approaches. We show that with the new cognitive architecture gives structure and transparency, suggesting that the use of LLMs offer the flexibility needed to support a variety of disassembly tasks. Furthermore, we used a VLM to enable human explainability of the visual instructions provided to the architecture in the form of exploded CAD diagrams. From these exploded CAD diagrams, ACT-FLEX was able to generate parametrised actions to perform a range of required tasks. This approach offers novices opportunities to use the architecture with very minimal technical expertise required. Though we did not use the knowledge library module, it provides an additional opportunity for people to embed regulations and other rules to be followed towards

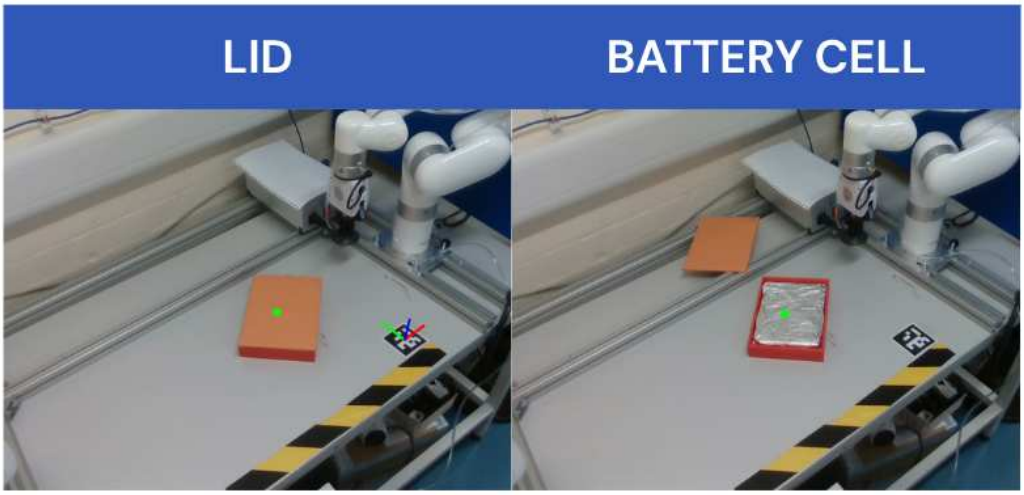

Figure 11: xArm at two different stages

Trial #	Success?	Reason of Failure
1	False	Failed to suck up the battery cell
2	True	NA
3	True	NA
4	True	NA
5	True	NA
6	True	NA
7	True	NA
8	True	NA
9	False	Failed to suck up the battery cell
10	True	NA

Table 10: Performance evaluation of the xArm in battery disassembly using boolean validity as a measure of success. The reasons for failure or success are presented in the column on the right.

completing a task.

Across the ACT-FLEX architecture, four prompts templates were used. However, we had to augment the results provided by the VLM due to it detecting the edge of an object instead of its centre. Towards this, we made use of a subagent system that included translation to Mandarin since Qwen2.5-VL, the VLM used, was a Chinese model. As a result, in total, we had to

use seven prompt calls.

A novelty about our architecture is that we did not need to train the Language Models used in order to conduct a variety of tasks across different robot morphologies. This large scale arrangement of a large number of Language Models in a single psychology inspired architecture is a first according to our present knowledge. It shows that robot cognitive flexibility can be achieved using this approach. However, there are current limitations with our approach which are starting points for future work.

Limitations: In terms of limitations, our current pipeline only considers the use of one high-level motion for the robots which in our case was the "pick-and-place" action. In future work, we will consider adding more high-level predefined motions to further support the generalisation of our architecture.

Another limitation in our pipeline is the absence of checking if each step has been successfully completed by an agent. In future we will implement a visual agent that provides an oversight on all the actions carried out in each step. Another future work is using the knowledge library to provide additional documentation and investigate how this impacts our results. This would also include adding regulation documents on the conditions of various components and to check whether they should be re-manufactured or scrapped.

In terms of morphological differences across robots, unlike the KUKA iiwa and Panda Arm, the UR10's second articulated joint has a non-zero offset. This means that the articulation is pushed outboard of the arm profile. As a result, the second joint sometimes induces a collision with the battery leaf which results in the robot dropping the base as well as in difficulty when picking up the battery cells. These limitations are consistent with our results that show that the UR10 scored lower most of the time on our metrics (See tables 6 to 8) when compared with the KUKA and Panda robots.

6. Conclusion

The development of personalised products to meet consumers' needs will result in automation challenges when it comes to recycling products en-masse on a common manufacturing line. This will lead to challenges for recycling and sustainability initiatives in the future, forcing the need for new automation methods. For automation to be effectively used, robots need to be capable of being rapidly reprogrammed and retasked. This calls for new reasoning modules that are capable of handling variations in product conditions

and differences in product configurations across various manufacturers.

Towards addressing this challenge, we present a new cognitive architecture called Adaptive Control of Thought and Rational with FLEXibility across Robot tasks and morphologies (ACT-FLEX) that was inspired by the informational processing pathway in humans. The architecture provided us with a framework within which we deployed a number of language models in order to bridge the gap between symbolic and neural emergent approaches. In particular, we made use of both Vision Language and Large Language Models to enable access to a wide range of knowledge. We showed that this approach enabled flexible decision making across different robotic platforms while being capable of disassembling various objects. As a result, this approach has the capability of being transferable to different robot morphologies thereby setting the foundations for a common architecture across various robot platforms.

Furthermore, we showed that our architecture can respond successfully to variations in object positions and orientations. Nevertheless, we have not yet implemented a verification and correction capability to check each step's completion. Addressing this limitation will require visual feedback validation techniques which we will investigate in the future.

Looking forward, we envision extending this work through more realistic EV battery disassembly as well as introducing more flexibility with tool changing. Overall, this research contributes an important step toward the long-term goal of achieving safe and trustworthy re-manufacturing robotic systems.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council under grants: A digital Cognitive architecture to achieve Rapid Task programming and flexibility in manufacturing robots through human demonstrations (DIGI-CORTEX) (EP/W014688/2) as well as the A*STAR research council Singapore.

References

[1] C. A. Bakker, F. Wang, J. Huisman, M. C. den Hollander, Products that go round: Exploring product life extension through design, Journal of Cleaner Production 69 (2014) 10–16. doi:10.1016/j.jclepro.2014.01.028.

[2] A. Ramji, H. Tayarani, A regional analysis of electric ldv portfolio choices by vehicle manufacturers, arXiv preprint arXiv:2307.03808 (2023). Preprint.

[3] T. Malloy, C. Gonzalez, Applying generative artificial intelligence to cognitive models of decision making, Frontiers in Psychology 15 (2024) 1387948.

[4] I. Kotseruba, J. K. Tsotsos, 40 years of cognitive architectures: core cognitive abilities and practical applications, Artificial Intelligence Review 53 (2020) 17–94.

[5] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al., A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, ACM Transactions on Information Systems 43 (2025) 1–55.

[6] C. Chakraborty, M. Bhattacharya, S. Pal, S. Chatterjee, A. Das, S.-S. Lee, Ai-enabled language models (lms) to large language models (llms) and multimodal large language models (mllms) in drug discovery and development, Journal of Advanced Research (2025).

[7] S. Wu, A. Oltramari, J. Francis, C. L. Giles, F. E. Ritter, Cognitive llms: Toward human-like artificial intelligence by integrating cognitive architectures and large language models for manufacturing decision-making, Neurosymbolic Artificial Intelligence 1 (2025) 29498732251377341.

[8] L. Xia, J. Pang, C. Li, R. Wang, P. Zheng, Large language models empower the reliability of disassembly in remanufacturing, Manufacturing Letters 41 (2024) 1728–1733.

[9] I. Díaz, D. Borro, O. Iparraguirre, M. Eizaguirre, F. A. Ricardo, N. Muñoz, J. J. Gil, Robotic system for automated disassembly of electronic waste: Unscrewing, Robotics and Computer-Integrated Manufacturing 95 (2025) 103032. URL: https://www.sciencedirect.com/science/article/pii/S0736584525000869. doi:https://doi.org/10.1016/j.rcim.2025.103032.

[10] Y. Zhou, Y. Peng, W. Li, D. T. Pham, Stackelberg model-based human-robot collaboration in removing screws for product remanufacturing, Robotics and Computer-Integrated Manufacturing 77 (2022) 102370.

[11] T. Wu, Z. Zhang, T. Yin, Y. Zhang, Multi-objective optimisation for cell-level disassembly of waste power battery modules in human-machine hybrid mode, Waste Management 144 (2022) 513–526.

[12] M. Choux, E. Marti Bigorra, I. Tyapin, Task planner for robotic disassembly of electric vehicle battery pack, Metals 11 (2021) 387.

[13] M. Zorn, C. Ionescu, D. Klohs, K. Zähl, N. Kisseler, A. Daldrup, S. Hams, Y. Zheng, C. Offermanns, S. Flamme, et al., An approach for automated disassembly of lithium-ion battery packs and high-quality recycling using computer vision, labeling, and material characterization, Recycling 7 (2022) 48.

[14] W. Zhang, Y. Li, K. Wang, W. Xu, L. Gao, A green and efficient disassembly line balancing with human-robot collaboration and destructive disassembly, Robotics and Computer-Integrated Manufacturing 97 (2026) 103081.

[15] W. Liang, Z. Zhang, Y. Zeng, D. Ji, Y. Zhang, H. Chen, Y. Li, L. Zhu, Modelling and optimization of mixed-parallel straight and two-sided disassembly line balancing problem, Swarm and Evolutionary Computation 97 (2025) 102045.

[16] J. Wang, M. Li, F. Meng, H. Zhao, X. Sun, Multi-objective reinforcement learning for dynamic balancing of two-sided human-robot collaborative disassembly lines, Computers & Industrial Engineering (2025) 111703.

[17] Y. Shen, W. Lu, H. Sheng, Y. Liu, G. Tian, H. Zhang, Z. Li, Human–robot collaboration on a disassembly-line balancing problem with an advanced multiobjective discrete bees algorithm, Symmetry 16 (2024) 794.

[18] J. Li, W. Qu, H. Zheng, R. Zhang, S. Liu, Generation of human–robot collaboration disassembly sequences for end-of-life lithium–ion batteries based on knowledge graph, AI EDAM 38 (2024) e13.

[19] S. Hjorth, D. Chrysostomou, Human–robot collaboration in industrial environments: A literature review on non-destructive disassembly, Robotics and Computer-Integrated Manufacturing 73 (2022) 102208.

[20] L. Xia, Y. Hu, J. Pang, X. Zhang, C. Liu, Leveraging large language models to empower bayesian networks for reliable human-robot collaborative disassembly sequence planning in remanufacturing, IEEE Transactions on Industrial Informatics (2025).

[21] J. Xiao, S. Terzi, Large language model-guided graph convolution network reasoning system for complex human-robot collaboration disassembly operations, Procedia CIRP 134 (2025) 43–48.

[22] J. Huang, X. Li, L. Gao, Q. Liu, Y. Teng, Automatic programming via large language models with population self-evolution for dynamic job shop scheduling problem, arXiv preprint arXiv:2410.22657 (2024).

[23] X. Guo, C. Jiao, P. Ji, J. Wang, S. Qin, B. Hu, L. Qi, X. Lang, Large language model-assisted reinforcement learning for hybrid disassembly line problem., Mathematics (2227-7390) 12 (2024).

[24] Y. Wan, Y. Liu, Z. Chen, C. Chen, X. Li, F. Hu, M. Packianather, Making knowledge graphs work for smart manufacturing: Research topics, applications and prospects, Journal of manufacturing systems 76 (2024) 103–132.

[25] J. E. Laird, The Soar Cognitive Architecture, MIT Press, 2012.

[26] J. R. Anderson, Act: A simple theory of complex cognition., American psychologist 51 (1996) 355.

[27] J. G. Trafton, L. M. Hiatt, A. M. Harrison, F. P. Tamborello, S. S. Khemlani, A. C. Schultz, Act-r/e: An embodied cognitive architecture for human-robot interaction, Journal of Human-Robot Interaction 2 (2013) 30–55.

[28] S. Wu, A. Oltramari, J. Francis, C. L. Giles, F. E. Ritter, Cognitive llms: Toward human-like artificial intelligence by integrating cognitive architectures and large language models for manufacturing decision-making, Neurosymbolic Artificial Intelligence (2024).

[29] A. Lieto, F. Perrone, G. L. Pozzato, E. Chiodino, Beyond subgoaling: A dynamic knowledge generation framework for creative problem solving in cognitive architectures, Cognitive Systems Research 58 (2019) 305–316.

[30] J. E. Laird, R. E. Wray, S. Jones, J. R. Kirk, P. Lindes, Proposal for cognitive architecture and transformer integration: online learning from agent experience, in: Proceedings of the AAAI Symposium Series, volume 2, 2023, pp. 302–306.

[31] S. Wu, A. Oltramari, J. Francis, C. L. Giles, F. E. Ritter, Cognitive llms: Toward human-like artificial intelligence by integrating cognitive architectures and large language models for manufacturing decision-making, Neurosymbolic Artificial Intelligence (2024).

[32] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288 (2023).

[33] R. Wray, C. Lebiere, P. Weinstein, K. Jha, J. Springer, T. Belding, B. Best, V. Parunak, Towards a complete, multi-level cognitive architecture, in: Proc. of the International Conference for Cognitive Modeling, 2007.

[34] H. Smieszek, N. Rußwinkel, Micro-cognition and macro-cognition: trying to bridge the gap, in: Proceedings of the 10th Berlin Workshop on Human-Machine Systems: Foundations and Applications of Human-Machine Interaction, 2013, pp. 335–341.

[35] H. Van Dyke Parunak, R. Rohwer, T. C. Belding, S. Brueckner, Dynamic decentralized any-time hierarchical clustering, in: International Workshop on Engineering Self-Organising Applications, Springer, 2006, pp. 66–81.

[36] J. R. Anderson, M. Matessa, C. Lebiere, Act-r: A theory of higher level cognition and its relation to visual attention, Human–Computer Interaction 12 (1997) 439–462.

[37] N. M. DiFilippo, M. K. Jouaneh, Using the soar cognitive architecture to remove screws from different laptop models, IEEE Transactions on Automation Science and Engineering 16 (2019) 767–780.

[38] A. Newell, Unified Theories of Cognition, Harvard University Press, 1990.

[39] J. E. Laird, C. Lebiere, P. S. Rosenbloom, A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics, AI Magazine 38 (2017) 13–26.

[40] J. A. H. Ali, M. Lezoche, H. Panetto, Y. Naudet, B. Gaffinet, Cognitive architecture for cognitive cyber-physical systems, IFAC-PapersOnLine 58 (2024) 1180–1185.

[41] J. E. Laird, The Soar cognitive architecture, MIT press, 2019.

[42] S. Wu, R. F. Souza, F. E. Ritter, W. T. Lima Jr, Comparing llms for prompt-enhanced act-r and soar model development: A case study in cognitive simulation, in: Proceedings of the AAAI Symposium Series, volume 2, 2023, pp. 422–427.

[43] J. N. Marewski, K. Mehlhorn, Using the act-r architecture to specify 39 quantitative process models of decision making, Judgment and Decision making 6 (2011) 439–519.

[44] J. Oyekan, C. Turner, M. Bax, E. Graf, From ontologies to knowledge augmented large language models for automation: A decision-making guidance for achieving human–robot collaboration in industry 5.0, Computers in industry 171 (2025) 104329.

[45] Y. Chen, C. Wang, L. Fei-Fei, C. K. Liu, Sequential dexterity: Chaining dexterous policies for long-horizon manipulation, arXiv preprint arXiv:2309.00987 (2023).

[46] V. C. Rodrigues, M. M. Kasaei, E. A. Marques, R. J. Carbas, L. F. da Silva, Adhesive bonding in automotive battery pack manufacturing and dismantling: a review, Discover Mechanical Engineering 4 (2025) 25.

[47] Mistral AI, Mistral small 3.1, `https://mistral.ai/news/mistral-small-3-1`, 2025.

[48] S. Bai, K. Chen, X. Liu, J. Wang, et al., Qwen2.5-vl technical report, 2025. URL: `https://arxiv.org/abs/2502.13923`. `arXiv:2502.13923`.

[49] Gemma Team, Gemma 3 technical report, 2025. URL: https://arxiv.org/abs/2503.19786. arXiv:2503.19786.

[50] Meta AI, Llama 3.2: Revolutionizing edge ai and vision with open, efficient models, https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, 2024.

[51] H. Liu, C. Li, Q. Wu, Y. J. Lee, Visual instruction tuning, in: NeurIPS, 2023.

[52] Y. Yao, T. Yu, A. Zhang, C. Wang, J. Cui, H. Zhu, T. Cai, H. Li, W. Zhao, Z. He, et al., Minicpm-v: A gpt-4v level mllm on your phone, 2024. URL: https://arxiv.org/abs/2408.01800. arXiv:2408.01800.

[53] I. A. Şucan, M. Moll, L. E. Kavraki, The Open Motion Planning Library, IEEE Robotics & Automation Magazine 19 (2012) 72–82. doi:10.1109/MRA.2012.2205651, https://ompl.kavrakilab.org.

Appendix A.

Appendix A.1. Large Language Model Prompts

As discussed earlier, in Figure 1, we used a VLM in module ①. This VLM defines the disassembly steps from the CAD images provided. Its prompt has one parameter `simple_task` which defines the task in a single sentence. The prompt is as follows:

```
Task: {simple_task}

Analyse the visual instruction image and Write a step-by-step
disassembly plan that includes only physical actions needed to
take apart the structure.

Instructions:

For each component, list it individually under components.

For each component, provide one matching disassembly step
under disassembly_steps, following the same order.

You must also give the COLORS of each components

The output must be a one-to-one correspondence
between components and actions.

Only include physical actions|do not include assessments,
observations, safety notes, or generic instructions.

Avoid non-actionable language such as "check,"
"assess," "observe," or "ensure."

If a tool is visible or clearly implied, list it with the step.

If something is unclear, infer the most probable physical
action and state it as a direct action.
```

```
Please check if the disassembly plan respects
the physical constraints:

    1. Bolts and fasteners must be removed before r
       emoving the parts they secure.
    2. Components on top must be removed before those below.
    3. Note: Do not include the base in the disassembly
       if nothing is physically attached to it or resting on it.
    4. Make sure you put the first actions respecting the
       physical constraints as first actions
       in the disassembly_steps.
```

The output of the VLM is structured using the following schema. Note that the `disassembly_steps` field will be looped and each of the array elements will be processed.

```
{
    "title": {
        "type": "string" ,
        "required": "true"
    },
    "components": {
      "type": "array",
      "items": { "type": "string" },
      "required": "true"
    },
    "pre_disassembly_checklist": {
      "type": "array",
      "items": { "type": "string" },
      "required": "false"

    },
    "disassembly_steps": {
      "type": "array",
      "items": { "type": "string" },
      "required": "true"
```

```
    },

    "safety_notes": {
      "type": "array",
      "items": { "type": "string" },
      "required": "false"
    }
}
```

In module ④, we made use of LLM to find the corresponding component in each task step (element in `disassembly_steps`). The current task is given as `action` parameter.

```
You are an expert in extracting objects and their roles
in mechanical or assembly tasks.

Your job is to read an instruction step and identify ONE
relevant physical objects mentioned, along with the action it
is involved in.
If color or type is mentioned, include that.

Use this schema:

- name: the full phrase used to describe
  the object (e.g., "pink block", "blue leg")
- color: the color if available (e.g., "pink", "blue"),
  otherwise leave null
- type: the general type of the object if mentioned
  (e.g., "block", "screw", "leg"), otherwise leave null
- action: the verb that applies to the object
  (e.g., "unscrew", "lift", "remove", "separate")

Now extract objects from this instruction:

#####        {action}          #######

return as JSON
```

The relevant object's description is given in the following schema.

```
{
    "name": {
        "type": "string" ,
        "required": "true"
    },
    "color": {
        "type": "string" ,
        "required": "false"
    },
    "type": {
        "type": "string" ,
        "required": "false"
    },
    "action": {
        "type": "string" ,
        "required": "false"
    }
}
```

After the name of the component is known, the pipeline calls a VLM in module ⑤ to get a 2-dimensional key-point on the image frame. Its prompt takes as a parameter the description of the object extracted above: `object_description`

```
You are a robot using the joint control, and vaccum gripper.
The task is "remove the {object_description}". Please predict
a possible affordance area of the end effector.
You must aim the center, not the countours

Return a JSON object with the following fields:

- "point_2d": The estimated 2D coordinates of the keypoint
  representing the object's affordance.

• If the object is a bolt, this should be the center of the bolt
```

```
  head.
- "label": The name or class of the object (e.g., "bolt",
  "screwdriver").

Only return a JSON object, no explanation.
```

The output follows the scheme as shown below:

```
{
    "point_2d": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "minItems": 2,
      "maxItems": 2,
      "required": true
    }
    "label": {
        "type": "string" ,
        "required": "true"
    }
}
```

Once the 2D key-point is predicted, the pipeline uses a topology algorithm and the ArUco marker to determine the object's 3D position in the environment.

The position is used in the `object_positions` parameter and the action from the step defined in the first VLM call as `action` in the prompt to module ⑦. The prompt specifies two primitives of actions: `pick_and_place` and `drag`. The drag action is not used in the experiments, and is discouraged to use in the text prompt. The reason is that we firstly experimented with a 2-finger gripper. The 2-finger gripper has a maximum width, and therefore was very difficult to determine that pose and orientation for best use. For instance, to lift the lid, we would need a specific orientation and position (edge of the lid) to perform the action, unlike a vacuum gripper where it can be placed anywhere on the lid's surface. Thus the drag action was useful to

push the components when it was too big, instead of gasping it. However, we stopped using the 2-finger gripper, therefore, we stopped using the drag action, but did not removed completely, we only discouraged it. The safe position is directly defined in the prompt as `x: 0.0, y: 0.7, z: 0.5` which is separated from the assembled initial position for our particular experiments on the 3 simulated robots in CoppeliaSim.

```
You are an expert in translating high-level manufacturing
instructions into low-level robot actions.

You can only choose 2 primitive action:

1) Pick and Place for fasteners, bolts and screws, and blocks,
and cylinders
2) Drag (DO NOT USE THAT UNLESS EXPLICITELY SAID TO) --
NEVER USE THAT FOR SLIDING, use the PICK AND PLACE

Here is the answer schema

object_id: str
move_from: Tuple[float, float, float]
move_to: Tuple[float, float, float]
description: str



You are given the instruction:
###### {action} ######

Your job is to:
    - Identify ONE object that needs to be moved.
    - Choose a safe location to move it from (move_from)
    and to (move_to).
    - Make reasonable guesses for positions if not given.
    - Write a short description of the action.
    - Always use the following coordinates as the safe
    deposit location (move_to):
```

```
    x: 0.0, y: 0.7, z: 0.5


If the instruction involves unscrewing or detaching, assume that
has already been done
you only need to perform the final lift/move.


Now generate the action for:
###### {action} ######

Object positions:
{object_positions}

Respond in JSON format.
```

The output follows the scheme as shown below:

```
{
    "type": {
        "type": "string" ,
        "enum": ['pick_and_place', 'drag']
        "required": "true"
    }
    "object_id": {
        "type": "string" ,
        "required": "true"
    }
    "move_from": {
      "type": "array",
      "items": {
        "type": "float"
      },
      "minItems": 3,
      "maxItems": 3,
      "required": true
    },
```

```
    "move_to": {
      "type": "array",
      "items": {
        "type": "float"
      },
      "minItems": 3,
      "maxItems": 3,
      "required": true
    }
    "description": {
        "type": "string" ,
        "required": "true"
    }
}
```

Declaration of Interest:

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.