

Nazarbayev University

School of Sciences and Humanities
Department of Computer Science

CSCI 409: Senior Project II

Final Project Report — Spring 2026

APRIL Robots Offline System: Microservices, Local Curriculum Execution, and Analytics

Team Members

Temirlan Yessembayev

Miras Akitov

Miras Omarov

Alibek Abilmazhit

Damir Naurzov

Advisers

Prof. Anara Sandygulova

Prof. Marat Isteleyev

Astana, Kazakhstan

April 2026

Table of Contents

Table of Contents	2
1. Executive Summary	4
2. Introduction	4
2.1 Problem Statement and Motivation	4
2.2 Proposed Solution	5
2.3 Report Organization	5
3. Identification of Constraints	5
3.1 Economic Constraints	5
3.2 Environmental and Operational Constraints	6
3.3 Social Constraints	6
3.4 Ethical Constraints	6
3.5 Legal and Regulatory Constraints	6
3.6 Health and Safety Constraints	6
3.7 Sustainability and Maintainability Constraints	6
4. Background and Related Work	7
4.1 Social Robotics in ASD Therapy	7
4.2 Furhat Robotics Platform and API Migration	7
4.3 ABLLS-R Skill Framework	7
4.4 Offline-First Architecture in Healthcare	8
5. Project Approach	8
5.1 System Architecture Overview	8
5.2 Backend and Data Model	8
5.3 ABLLS-R Skill Hierarchy and Evaluation	8
5.4 Robot Microservices: Unified Architecture	9
5.5 Furhat Microservice: Observation Pipeline	9
5.6 NAO Microservice: Observation Pipeline	9
5.7 Frontend: SvelteKit, Svelte 5 Runes, and Chart.js	10
5.8 Synchronization and Installer	10
5.9 Third-Party Components	10
6. Project Execution	11
6.1 Fall 2025: Foundations	11
6.2 Spring 2026: Completion and Validation	11
6.3 Team Roles and Collaboration	13
7. Evaluation	14
7.1 Evaluation Strategy	14
7.2 Automated Testing	14
7.3 ABLLS-R Skill Hierarchy Refinement	14
7.4 Public Demonstration	14

7.5 Connection Stability Testing	15
7.6 Limitations	15
8. Conclusion and Possible Future Work	15
8.1 Key Findings and Contributions	15
8.2 Future Enhancements	15
9. References	16

1. Executive Summary

The APRIL Robots Offline System is an independent Windows system that allows robot-assisted autism spectrum disorder (ASD) therapy where the internet is not consistently available or is limited. It enables experts in the field of therapy to set up and participate in sessions with the social robots (NAO and Furhat), gather behavioral measurements, and align data with the central APRIL online server (`api.robots.aprilhub.kz`) upon the connection reestablishment. This system uses a hierarchical skill taxonomy, which is developed according to the ABLLS-R (Assessment of Basic Language and Learning Skills - Revised) framework, and allows tracking development of each child in a structured way.

The project was a two-semester project. In Fall 2025, the team defined the core architecture in four repositories: the general API backend, Furhat robot microservice, NAO robot microservice, and SvelteKit/Electron frontend. The main deliverables were FastAPI microservices to support both robots, automated CI/CD pipelines, complete upstream synchronization with the online backend, and an automated Windows installer.

In Spring 2026, five major development pillars were completed:

- **Metrics Collection Pipeline:** The Furhat microservice has been adapted to include the event loop architecture of an observer to capture real-time frame captures and stream user detections and include the CV model inference layer into the pipeline.
- **Complete Microservices:** The backend's test suite reached 26 test functions across 6 modules. The Furhat microservice has around 15 endpoint tests including connection management, action lifecycle, queue state and battery status.
- **Frontend Improvements:** The metrics visualization dashboard has been introduced, which includes doughnut charts and stacked bar charts to get longitudinal session analytics. The multi-robot control panel enables the specialists to alternate between Furhat and NAO, battery monitoring and volume control of NAO.
- **System Reliability:** Connection stability testing, a better offline application behavior, a structured error handling system, and database migration system were introduced throughout the backend.
- **Pilot Studies and Validation:** Expert evaluation was conducted, including refinement of the ABLLS-R-based skill hierarchy for child development. A public demonstration was held for the target audience. Full data analysis and refinement could not be completed due to insufficient data volume.

2. Introduction

2.1 Problem Statement and Motivation

ASD children find such structured, repetitive and engaging therapeutic interactions of great benefit. Social robots have been found successful in ASD therapy because of the predictability, consistency and non-judgemental behaviour. In Kazakhstan, the APRIL (Adaptive Personalized Robot-Assisted Instruction and Learning) research group at Nazarbayev University has been exploring the use of social robots to treat ASD, using NAO (SoftBank Robotics) and Furhat (Furhat Robotics) platforms.

In therapy centers in Kazakhstan, there is often unstable or slow internet access, and cloud-based solutions are unrealistic. Privacy, consent, and data protection privacy and protection issues should also be carefully addressed in collecting and analyzing child behavioral data during sessions in accordance with the Kazakhstani law. Monitoring child development across time requires a systematic measurement system featuring stable, clinically based skill taxonomies.

2.2 Proposed Solution

This project develops the APRIL Robots Offline System: a standalone Windows application organized into four repositories. The backend is a FastAPI application that stores data locally using a JSON-file-based store and synchronizes with the APRIL online platform. There are two robot microservices. They run as containerized services inside a Podman environment managed by the installer. The frontend is a SvelteKit 2 application using Svelte 5 Runes packaged as a desktop app via Electron.

The system organizes any content of the therapy based on an ABLLS-R-based hierarchy of skills (SkillArea → SkillDomain → Skill) enabling specialists to monitor child progress according to standard developmental levels. The data of the sessions, profiles of children, and metrics is stored on the computer using nine JSON collections and is synchronized to the central server when connected to the internet. The files are synchronized with the robots via SSH file uploading where therapy audio and behavior scripts are uploaded directly to the robot.

2.3 Report Organization

The remainder of this report is organized as follows. Section 3 identifies the constraints shaping the project. Section 4 reviews related work. Section 5 describes the system architecture. Section 6 details project execution across both semesters. Section 7 presents evaluation results. Section 8 concludes with findings and future directions. References follow in Section 9.

3. Identification of Constraints

The APRIL Robots Offline System design and implementation were influenced by various constraints in the economic, environmental, social, ethical, legal, health and safety, and sustainability aspects.

3.1 Economic Constraints

The NAO and Furhat robots are expensive commercial hardware, and cost several thousand to tens of thousands of dollars each. The team used solely the existing inventory at the university which limited concurrent multi-robot testing. Software dependencies were all chosen either out of open-source or freely available packages to keep costs down. Containerization using Podman instead of Docker avoids any issues related to licensing in commercial deployments.

3.2 Environmental and Operational Constraints

The main limitation of operation is the restricted or intermittent internet connection in therapeutic facilities. This motivated the offline-first design: any local collections of JSON should be complete and should not require connectivity. The application should be able to install and run on standard Windows machines without special IT assistance, which predetermines the automated installer pipeline based on PowerShell and Inno Setup.

3.3 Social Constraints

The system targets a vulnerable group of children with ASD and their families. The interface should not confuse specialists who might have low technical skills. SvelteKit frontend is entirely translated into Kazakh, Russian, and English with the help of YAML translation files to make sure that it is accessible to the multilingual therapy workforce in Kazakhstan.

3.4 Ethical Constraints

The system gathers confidential information regarding minors, such as behavioral indicators and developmental evaluations. The most important ethical limitations are: seek informed consent with legal guardians prior to developing child profiles; gathering data just sufficient to therapy; making certain that recommendations based on metrics do not take precedence over the therapist's judgment. The API implements role-based access where parents, specialists and administrators can only access data relevant to their roles.

3.5 Legal and Regulatory Constraints

The system should be in adherence with the Law of the Republic of Kazakhstan No. 94-V "On Personal Data and Its Protection" (2013). Given that child data is sensitive data under this law, the following requirements are added: child records can only be established with clear consent of the guardian. Licenses of open source components (MIT, Apache 2.0) are honored in distribution. GitHub Container Registry (ghcr.io) is used to build the production and all secrets are stored in GitHub Actions secrets.

3.6 Health and Safety Constraints

It is essential to have a reliable system when active therapy is conducted: even a sudden change in the behavior of a robot or the crash of an application can cause stress to a child with ASD. Each microservice queue manager separates action execution in dissimilar multiprocessing environments such that failure in action execution in one service will not cause crash of the service. The exponential backoff logic of the retry logic tries to overcome the temporary connection failures before stopping. Docker Compose configurations of robot microservices have infinite-retry health checks so that the services are always available to the backend.

3.7 Sustainability and Maintainability Constraints

Each repository has a FRAMEWORK.md describing the modular microservice architecture in which universal components (QueueManager, ScriptRunner, ObserverThread) are

segregated into robot-specific components (RobotInterface, ObserverInterface, metric_computation.py). This design enables each robot adapter to be updated separately. The backend (versioned database migration system v1.0.0, v1.0.1) is designed to make sure that local data stores can be non-destructively upgraded to software releases. The release pipeline is controlled by semantic versioning and documented GitFlow release process.

4. Background and Related Work

4.1 Social Robotics in ASD Therapy

The application of social robots in the treatment of ASD is not a new topic of study; it has been researched for more than 10 years. Platforms that have been most popularly deployed include NAO (SoftBank Robotics). Bekele et al. (2013) created the Adaptive Robot-Mediated Intervention Architecture (ARIA) based on NAO, with a network of cameras and sensor cap to track gaze and respond to it adaptively with robot gestures and verbal prompts. This article showed that closed-loop, sensor-based robot intervention can be used to treat ASD and set the standard that real-time behavioral sensing is the key to adaptive interactions.

The European DREAM project used NAO as the main platform, with the multimodal sensors of the robot to produce socially contingent responses in children with ASD. DREAM provided important insights to autonomous and semi-autonomous therapy delivery architectures, which informed the design of the session execution engine of the APRIL system.

4.2 Furhat Robotics Platform and API Migration

Furhat Robotics offers a conversational AI platform that has a back-projected face robot. In the Fall 2025 semester, the APRIL team released the first Furhat microservice that utilized FurhatRemoteAPI, which is a REST-like interface. Later Furhat Robotics deprecated FurhatRemoteAPI in favor of FurhatRealtimeAPI, a WebSocket-based real-time interface (AsyncFurhatClient). The migration required substantial refactoring of the observer, introducing a dedicated event loop thread that runs alongside the main FastAPI ASGI loop. The FurhatRealtimeAPI allows to stream user detection events and camera frames at specified frequency, which is vital to the real-time observation pipeline.

4.3 ABLLS-R Skill Framework

The Assessment of Basic Language and Learning Skills—Revised (ABLLS-R) is a widely used criterion-referenced assessment tool for children with developmental disabilities (Rababa & Ayasrah, 2024). It organizes skills into a hierarchical structure of domains and sub-skills, providing standardized benchmarks for tracking progress. The APRIL Offline System adopts this three-level hierarchy (SkillArea → SkillDomain → Skill) as its curriculum organization framework. The 170+ therapy action scripts in the NAO microservice (organized into Kazakh and Russian sub-directories across domains including songs, emotions, dances, fairy tales, body parts, imitations, social skills, and logorhythmics) are mapped to skills in this taxonomy, enabling systematic developmental tracking.

4.4 Offline-First Architecture in Healthcare

The majority of current robot-assisted therapy platforms presuppose the presence of reliable internet connectivity to manage the sessions and infer cloud AI (Saha & Dasgupta, 2018). APRIL system closes this gap by operating an offline-first system with a local dictdatabase JSON store with a synchronization module uploading anonymized session metrics and downloading updated content on reconnect. File synchronization Paramiko-based SSH is used to keep robot audio resources and behavior scripts up to date with the central server. This approach aligns with broader trends toward edge computing in healthcare, where local processing is necessary for data privacy and operational reliability.

5. Project Approach

5.1 System Architecture Overview

The APRIL Robots Offline System consists of four repositories:

- april-robots-offline-general-api: main FastAPI server, database, synchronization service, session assessor system, and Windows package pipeline.
- april-robots-agent-furhat: Microservice in a container to control Furhat robots, stream observations, and compute metrics.
- april-robots-agent-nao: dockerized microservice that controls NAO robots, gathers data through NAOqi, and calculates metrics.
- april-robots-offline-fe: SvelteKit 2 application, using Electron as a desktop app.

The backend connects to both robot microservices via an async HTTP (aiohttp). The robot microservices return completion data on the action-completed endpoint of the backend, with computed metrics and raw observation data. The frontend is in contact with the backend via a locally served REST API.

5.2 Backend and Data Model

All data is stored locally in the backend (FastAPI 0.116.1, Python 3.12) in a file-based JSON storage library (dictdatabase). This allows us to run a database without server dependency, making the installer self-contained. The schema updates in each release are applied in a non-destructive way by a version aware migration system (database/migrations/).

In the data model, the following main entities are involved: child, session, performedAction, skill, and assessmentQuestion. Records have ObjectId identifiers and created at times in UTC.

5.3 ABLLS-R Skill Hierarchy and Evaluation

The contents in the therapy are arranged as a three-level taxonomy of ABLLS-R, the latest addition: SkillArea (top, color-coded), SkillDomain (mid, connected to one or more areas), and Skill (atomic ABLLS-R unit with robot measures). Every step of the system is connected to some skills, as the mapping that was created by a psychology specialist at our lab illustrates. Specialists apply the SpecialistGrade model to record grades on a 1-10 scale on each corresponding skill after each session. The AssessmentQuestion system is used to gather post-session assessment questions.

5.4 Robot Microservices: Unified Architecture

Robot microservices are based on a common architecture. The universal layer is composed of QueueManager (action lifecycle management), a ScriptRunner/Service (scripts called as a subprocess with a retriability mechanism) and ObserverThread (abstract threading). Thread (observation collection thread), and ProcessManager (multiprocessing context shared observation state singleton). Robot-specific layer includes RobotInterface (SDK wrapper), ObserverInterface (polling of sensors), operations.py (connection, battery, volume), and metric-computation.py (calculation of engagement score).

The execution of action is a multiprocessing isolation pattern. Upon a request of the backend, the microservice creates a new process. Action script module is dynamically loaded and its `run(robot_interface)` function, which is an asynchronous function, is invoked. Gathering behavioral information in a common dict is done by running concurrently subprocesses called ObserverThread. Upon completion, the dict is finalized into an ObservationsData object and metrics are calculated and a callback HTTP POST is sent to the backend.

5.5 Furhat Microservice: Observation Pipeline

Both FurhatRemoteAPIs (to permit calls to robot control) and FurhatRealtimeAPIs (to permit real-time WebSocket streaming) are used by the Furhat microservice. The ObserverInterface has its own event loop thread together with the FastAPI ASGI event loop. Frames of captured images (base64 JPEG) and user detection data are added to the queue of jobs that are processed by the main observer thread.

The Furhat Microservice is built on an event-driven ObserverInterface architecture that uses asynchronous communication to achieve real-time engagement tracking. The pipeline consumes JPEG frames and spatial user position data at a 1-second rate by registering on-users-data and on-camera-data handlers, which read the data. They are sent out as observation jobs to a decoupled inference layer, and thus non-vital vision processing like 3D gaze estimation or object detection do not block the main robot control cycle or state transitions.

5.6 NAO Microservice: Observation Pipeline

The NAO microservice is controlled with the NAOqi SDK (qi library) to manipulate robot behaviors mounted on the physical robot (port 9559). The ObserverInterface is a subscriber to the events of arrival and departure of people in PeoplePerception. Upon detection of someone, it samples two NAOqi modules in each observation time (1.0 s): ALMood (attention, valence, smile, facial expressions, gaze score, body language ease), and ALMemory PeoplePerception keys (position in robot/torso/world frame, distance, is face detected, is looking at robot, engagement zone, shirt color). Two aggregate measures are calculated: (1) attention: mean of ALMood attention scores, in which confidence = 0.7 or above; and (2) engagement: 0.4 is looking at robot + 0.3 gaze score + 0.2 zone proximity + 0.1 valence.

The NAO microservice contains 170+ action scripts grouped into a lib/ directory of reusable pose/gesture animation, and domain-specific subdirectories (kk/ Kazakh, ru/ Russian) with directions such as songs, emotions, dances, fairy tales, body parts, imitations, social skills, and logorhythms. All action scripts adhere to a standard contract: a `run(robot_interface:`

RobotInterface) function which interacts with the RobotInterface wrapper to make NAOqi calls like angleInterpolationBezier, start behavior, play audio, wait-for-touch.

5.7 Frontend: SvelteKit, Svelte 5 Runes, and Chart.js

The front end is developed using SvelteKit 2.17.2 and Svelte 5.20.4, which is reactive state management based on the new Runes API but no longer uses traditional stores. The program is bundled as an Electron desktop application and a SvelteKit static adapter produces the production build.

The primary navigation links include: /curriculum (browse activities by category and skill), /exercises (session control panel at the multi-robot therapy), /children (child management and profile dashboards), /sessions (session history and evaluation), /skills (ABLLS-R hierarchy browser) and /help. The child profile dashboard shows Chart.js doughnut charts (Radial.svelte) and stacked bar charts (StackedBar.svelte) of session duration analytics. The NAO battery level has an 8 icon visual indicator, and the NAO volume is controllable (0-100) through a slider.

The frontend is completely localized to Kazakh (kk), Russian (ru), and English (en). Session inactivity monitoring: This is an integration that monitors session inactivity and sends a keepalive to the backend using configurable thresholds.

5.8 Synchronization and Installer

The sync module of the backend controls cloud synchronization. Upon triggering, it gathers all local sessions and children, uploads them to the APRIL online platform with JWT Bearer token and downloads new actions, categories, skills, questions, metrics, and fixed images. To synchronize robots, an SSH client is used to upload absent audio resources and behavior scripts directly to robot filesystems by SFTP.

The windows installer is created with PyInstaller and Electron and orchestrated by a PowerShell build script. Installing the Inno Setup based installer silently sets up WSL, Virtual machine platform and podman during installation, starts the podman machine to run the robot microservice containers and injects the JWT secret into the application .env file. The user-generated data (children.json, sessions.json) is maintained through updates.

5.9 Third-Party Components

- FastAPI 0.116.1 — backend REST API framework (Python)
- SvelteKit 2.17.2 / Svelte 5.20.4 — frontend framework with Runes reactivity
- Electron — desktop application wrapper for the SvelteKit frontend
- dictdatabase 2.5.1 — JSON-file-based local data store (offline storage)
- Chart.js 4.4.8 — doughnut and stacked bar chart visualizations
- Tailwind CSS 4.0.8 — utility-first styling
- FurhatRealtimeAPI / AsyncFurhatClient — WebSocket-based Furhat robot streaming API
- NAOqi (qi library) — NAO robot SDK (port 9559)
- paramiko 3.5.1 — SSH/SFTP file synchronization to robots
- aiohttp 3.13.3 — async HTTP client for microservice communication
- Podman — rootless container runtime for robot microservices on Windows

- PyInstaller 6.15.0 — Python-to-executable compilation for the backend
- GitHub Actions — CI/CD pipeline for testing, building, and releasing

6. Project Execution

6.1 Fall 2025: Foundations

In the Fall 2025 semester, the basic architecture was established in all four repositories. Key deliverables included:

- NAO and Furhat Microservices: We implemented session execution, connection management module handling retry logic, queueing of actions using `asyncio.Queue`, subprocess-based script isolation, and real-time observation collection. Overall, the four-component architecture (`QueueManager`, `ScriptRunner/Service`, `ObserverThread`, `ProcessManager`) was built and documented.
- Automated CI/CD Pipeline: GitHub Actions workflow was set up on the backend. Both microservice repositories for robots have distinct workflows to build and push Docker images to `ghcr.io` on pushes to main and release branches.
- Backend API and Test Suite: The 26-test suite of the backend was written in five modules (`test_sessions.py`, `test_children.py`, `test_actions.py`, `test_evaluation.py`, and `test_skills.py`). The system of the ABLLS-R was filled with skills hierarchy and the system of assessment questions and specialist grades were adopted.
- Full Upstream Synchronization: The sync module was introduced to push local data to the APRIL online platform and pull down updated data. The mechanism of checking the version was implemented. The baseline of the database migration system was v1.0.0.
- Offline App Installer: PowerShell build pipeline, PyInstaller specifications, and Inno Setup installer were created and tested, which automated WSL, Podman, and network settings without the involvement of IT.

A major issue in Fall 2025 was the fact that `FurhatRemoteAPI` was being deprecated and replaced by `FurhatRealtimeAPI`. The Furhat microservice was refactored to add a new `AsyncFurhatClient` with a separate event loop thread, `WebSocket` heartbeat control, and camera and user detection streaming event-driven. Although this migration slowed down delivery, it created a stronger base on real-time observation gathering.

6.2 Spring 2026: Completion and Validation

Spring 2026 was dedicated to finishing the metrics pipeline, increasing coverage and testing, developing the frontend, making the system more reliable, and testing it on real users and experts.

6.2.1 Metrics Collection Pipeline

For Furhat, the metrics collection pipeline is deployed as a multi-modal inference layer comprising three specialized scripts to extract high fidelity behavioral data. The original script, `combined_gaze_laser_1.0.py`, checks whether a user is gazing at the camera lens of the robot, based on MediaPipe PnP-based head pose estimation, iris tracking. It uses a 3D angular intersection test instead of 2D hitboxes to be spatially correct in user movement, and collects raw signals such as Gaze Angle (3D angular deviation), a 3D Gaze Vector (X, Y, Z line of sight),

Smoothed Yaw and Pitch (filtered via Exponential Moving Average to remove jitter), and a binary Engagement Status using a distance-scaled dynamic threshold

Eye_at_object-4.0.py (based on YOLOv8) is used to locate a Rubiks cube and estimate whether a projected gaze ray is intersecting with the Oriented Bounding Box (OBB) of the object in order to monitor interaction with physical objects. This step produces a Gaze Intersection hits metric, Iris Ratio (iris location in the eye socket) metric, Head Yaw to weight head-vs-eye contribution, YOLO Object Confidence scores and a detection status trigger. Lastly, emotion_recognition_enet_b0_afew.py offers facial emotion recognition (FER) with the hsemotion library and EfficientNet-B0 model, which is trained on the AFEW dataset. It takes tight MediaPipe face crops every 500 ms to produce the Dominant Emotion label, Emotion Confidence, real-time Processing FPS, and Face Proximity coordinates. These groupings are compiled in metric_computation.py to come up with five summary measures: face-detection, attention-to-robot, proximity, and proximity-based sitting/standing ratios.

In the case of NAO, the engagement scoring formula was completed: $0.4x \text{ is_looking_at_robot} + 0.3x \text{ gaze_score} + 0.2x \text{ engagement_zone_proximity} + 0.1x \text{ valence}$. Attention is computed as the average of ALMood attention values where the corresponding confidence value is at or above 0.7, filtering out low-confidence readings.

6.2.2 Test Coverage Expansion

The final backend test suite had 26 test functions in 5 modules, and coverage of all key API surface areas. The integration cases tested are session lifecycle (create, start robot session, record action grades, end session, retrieve), child management (CRUD including validation of required fields and invalid IDs), category and action retrieval, evaluation question and metric access, and skill hierarchy navigation.

The Furhat microservice has about 15 endpoint tests in tests/test_endpoints.py, which include: the health check endpoint, access control enforcement when no robot is connected, connection validation with invalid and valid IP addresses, robot disconnection, empty and populated queue state, battery level reporting (hardcoded 100 in Furhat, as it has no battery) and full status response structure, action enqueueing (single and multiple, verifying is_first flag), action stopping, current action lifecycle tracking (queued -> running -> stopped), and volume control.

The NAO microservice does not have any automated test files as of now; the CI pipeline installs Python 3.12 and moves on to the Docker image build without any tests being run. This has been recognized as one of the gaps that are partially resolved by the extensive documentation of the shared architecture and the backend integration tests that cover the NAO action callback path.

6.2.3 Frontend Enhancements

The child profile route was also implemented with a metrics visualization dashboard, which includes two types of chart.js charts. Frontend displays doughnut charts, including inner percentage and HH:MM:SS time labels, displaying per-robot use as a fraction of overall maximum session use. A Calendar component is date-range filtering of session analytics.

An eight-level icon set is used to show battery level, and is polled at a settable interval. NAO has a volume control implemented as a slider, which sends POST /nao/set-volume; the equivalent Furhat endpoint is also present but not implemented in the microservice. This can be attributed to the fact that the new Furhat SDK is still in its infancy and the team attempted to

contact the support of Furhat but received an email that this will be resolved in the subsequent releases of Furhat RealtimeAPI.

6.2.4 System Reliability

The stability of connections was enhanced on both robot microservices. The error handling of the backend was redesigned to give semantic HTTP response codes: HTTP 400 on invalid inputs, HTTP 403 on SSH authentication errors, HTTP 404 on missing resources, HTTP 502 on microservice errors and HTTP 426 on client version errors. The database migration system was extended to support schema updates of early Fall deployments. Connection management of the Furhat microservice was further extended with `CONN_TIMEOUT = 5 s` and controllable `MAX_CONNECT_RETRIES` with the connection operation invoking stop- operation (which halts all robot activities) prior to reporting connection status.

6.2.5 Pilot Studies and Expert Evaluation

Professional assessment was done in cooperation with ASD therapy personnel and the researchers of child development. The skill hierarchy was examined and optimized based on ABLLS-R: the mappings between skills and their actions were fixed, the description of skills was revised in all the three languages (Kazakh, Russian, English), and the structure of the radar graph on which the individual child skill progress was visualized was changed to reflect developmental sequences better.

The target audience (consisting of therapy specialists of partner centers and parents of children with ASD) was involved in a public demonstration. Experts feedback was used to validate the practical usefulness of offline-capable operation and metrics dashboard was found to be useful in session reporting. There were a few improvement ideas recorded such as multi-session trend charts and the possibility to annotate the events in the timeline of the session. Robots and the system were successfully displayed as part of the «Research and Development for Autism» conference, organised by Prof. Anara Sandygulova on April 18th at NU.

The refinement and data analysis phase could not be done. The amount of therapeutic sessions performed in Spring 2026 was not adequate to perform statistically significant longitudinal metrics analysis, or produce the training data needed by the intended RL engine. This is reported to be the main drawback of the present assessment.

6.3 Team Roles and Collaboration

The team of 5 members distributed the component ownership in four repositories. The weekly sprint planning meetings were conducted with the help of GitHub Projects where the tasks were assigned and the progress was monitored. Communication was held daily by means of Discords and Telegram. Code review was mandated by GitHub pull request, and pre-commit hooks (ruff to lint and format) were used to maintain a high code standard in the Furhat microservice repository. Joint integration sessions were used to coordinate the work of cross-component integration, as in complementing the schema output format of the observation schema output in both microservices with the `PerformedAction.metrics_raw_data` field of the backend.

7. Evaluation

7.1 Evaluation Strategy

The assessment was based on three questions: (1) Does the system consistently undertake offline therapy sessions with NAO and Furhat robots? (2) Does the engagement measurement, calculated, provide significant behavioral indicators that are clinically significant? (3) Can the specialist interface be used by non-technical practitioners in the therapy? The automated testing, connection stability testing, assessment of the skill hierarchy by human experts and a general demonstration with the feedback obtained were the testing methods.

7.2 Automated Testing

The backend test suite (26 functions in 5 modules) was run on each push to build and release branches through GitHub Actions CI. A temporary dictdatabase storage directory is used as the test environment, mongomock on any MongoDB-dependent paths, and TestClient (httpx) of FastAPI to test endpoints synchronously. The CI pipeline was tested, and all 26 tests passed, including session CRUD lifecycle, child management, activity retrieval, evaluation data, and access to the hierarchy of skills.

The Furhat microservice test suite (around 15 tests in test_endpoints.py) is executed with pytest-asyncio and asgi-lifespan to test the entire FastAPI application in an asynchronous test environment. Tests authenticate endpoint access control, connection validation, action lifecycle (enqueue, queue state, stop, current action tracking), battery reporting, and complete status response structure.

7.3 ABLLS-R Skill Hierarchy Refinement

The specialist assessment was much more concentrated on the ABLLS-R hierarchy of skills. The mapping of therapy action scripts into skills and gaps in skill sequencing and domain assignments were reviewed by the psychology specialist. After the assessment, the skill hierarchy was revised: some assignments of skills domains were fixed, the action-to-skill relations were changed and the translations in Kazakh were improved with the help of a native-speaking expert. The radar graph model in the child profile dashboard of the frontend was redesigned to fit with the updated domain boundaries.

7.4 Public Demonstration

An open test of the entire integrated system was done in the presence of therapy specialists and parents. The demonstration included: getting setup, connecting to Furhat and NAO (connecting to both Furhat and NAO via the /exercises panel), running a therapy session with real-time status feedback, reviewing session metrics in the child profile dashboard (Radial and StackedBar charts), navigating the ABLLS-R skill browser, and a cloud synchronization. Expert commentary demonstrated the usefulness of the offline-first mode with centers with questionable connectivity and the usefulness of the multi-robot control panel with centers that have varied robots by curriculum phase. Multi-session trend overlays and annotation of session events were in the feature request. Our project was later presented at a conference on autism research and met with favourable response by various experts in the field of ASD in recognising the robotherapy method of treating ASD patients.

7.5 Connection Stability Testing

Connection stability The connection stability was evaluated by interrupting robot network connections at different levels during ongoing sessions. The event loop reconnection logic of Furhat microservice and the MAX_CONNECT_RETRIES = 3 retry wrapper of the backend were tested on the following conditions. It was found that the system was able to recover disconnections in most of the test cases and was able to complete activities or gracefully terminate activities without the loss of data. Health checks of the robot microservice ensured that the Podman container restarted left sessions in an inconsistent state.

7.6 Limitations

Its main weakness is lack of enough volume of session data. Data analysis and refinement stage was incomplete as the number of therapy sessions in the Spring of 2026 was too low to statistically valid longitudinal metric analysis or to generate RL training data. Also, the NAO microservice does not have automated tests and the volume control feature of Furhat microservice is not implemented. They are high priority items to the next stage of development.

8. Conclusion and Possible Future Work

8.1 Key Findings and Contributions

APRIL Robots Offline System provides an offline platform that is production-quality and provides robot-assisted ASD therapy. The team built a total of four interconnected repositories during two semesters comprising: a FastAPI backend, an ABLLS-R three-level skill taxonomy, a upstream synchronization module, SSH-based robot file sync, a Windows installer; a Furhat microservice with five engagement metrics, and CI/CD Docker builds; a NAO microservice with two engagement metrics,

The main technical achievements are: a unified and extensible microservice architecture to abstract heterogeneous robot SDKs using a common interface, a weighted, clinically informed system of engagement metrics per type of robot, an offline-first data architecture with versioned migrations and cloud sync, and a completely automated deployment pipeline that allows the system to be installed on commodity Windows hardware without requiring IT knowledge.

8.2 Future Enhancements

- Data Analysis and Metric validation: The more therapy sessions are gathered, the more the engagement metrics have to be verified against clinician's behavioral annotations to determine the inter-rater reliability and clinical validity.
- Reinforcement Learning Engine: The next significant stage of development is the RL engine. A policy engine is observed by the metrics infrastructure (face_detection, attention_to_robot, proximity, attention, engagement) and the data of the session state. The queue manager and script runner provide the action execution substrate into which RL-generated action recommendations can be injected.
- QT and Moxie Microservices: The system architecture is capable of supporting more adapters to the robots by providing the RobotInterface, ObserverInterface, and

metric_computation.py contracts to each platform. QT and Moxie adapters will be in a future semester.

- Multi-Center Pilot Deployment: A formal multi-center pilot using uniform ASD outcome measurement tools would deliver quantitative data on the effects of the system on quality of therapy and child outcomes.

9. References

- Bekele, E. T., Lahiri, U., Swanson, A. R., Crittendon, J. A., Warren, Z. E., & Sarkar, N. (2013). A step towards developing adaptive robot-mediated intervention architecture (ARIA) for children with autism. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 21(2), 289–299.
- Furhat Robotics. (2023). Advancing autism therapy with robots — Si Chen's Linguistics Lab [Blog post]. Furhat Robotics.
<https://www.furhatrobotics.com/use-cases-and-concepts/improving-interactions-in-children-with-autism>
- Furhat Robotics. (n.d.). Furhat SDK. <https://www.furhatrobotics.com/furhat-sdk>
- Open Source DREAM Project. (n.d.). Robotic platform. Retrieved April 2026, from
<https://dream2020.github.io/DREAM/platform.html>
- Partington, J. W. (2006). Assessment of Basic Language and Learning Skills—Revised (ABLLS-R). Behavior Analysts.
- Rababa, M., & Ayasrah, S. (2024). The effectiveness of a training program based on the assessment of basic language and learning skills- revised tool 'ABLLS-R' in reducing stereotyped behaviors among children with autism spectrum disorder. *International Journal of Developmental Disabilities*, 70(6), 1068–1081.
<https://doi.org/10.1080/20473869.2024.2380942>
- Saha, O., & Dasgupta, P. (2018). A Comprehensive Survey of Recent Trends in Cloud Robotics Architectures and Applications. *Robotics*, 7(3), 47.
<https://doi.org/10.3390/robotics7030047> Cited by: 204
- Salhi, I., Qbadou, M., Gouraguine, S., Mansouri, K., Lytridis, C., & Kaburlasos, V. (2022). Towards robot-assisted therapy for children with autism — The ontological knowledge models and reinforcement learning-based algorithms. *Frontiers in Robotics and AI*, 9, Article 713964.
- SoftBank Robotics. (n.d.). NAO — The humanoid and programmable robot.
http://doc.aldebaran.com/2-1/home_ao.html
- The Law of the Republic of Kazakhstan No. 94-V "On Personal Data and Its Protection". (2013, May 21).