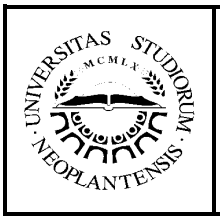


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Александар Ћетковић**
Број индекса: **РА 24/2021**

Тема рада: **Алат за помоћ у превођењу Структуираног текста у Пајтон коришћењем ЛЛМ Мистрал на платформи АВС Бедрок**

Ментор рада: **проф. др Мирослав Поповић**

Нови Сад, Јул, 2025

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Александар Ћетковић	
Ментор, **МН**:		проф. др Мирослав Поповић	
Наслов рада, **НР**:		Алат за помоћ у превођењу Структуираног текста у Пајтон коришћењем ЛЛМ Мистрал на платформи АВС Бедрок	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2025	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/30/6/8/11/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника и рачунарске комуникације	
Предметна одредница/Кључне речи, **ПО**:		транспајлер, ПЛЦ контролер, Структуирани Текст, Пајтон, АИ модел	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		У овом раду представљен је развој софтверског алата за помоћ у превођењу програма написаних у језику Структуирани Текст, који се користи за програмирање ПЛЦ контролера, у Пајтон код уз коришћење постојећих Пајтон библиотека. Уместо ручног мапирања, алат користи модел вештачке интелигенције за повезивање функцијских блокова и функција из СТ библиотека са одговарајућим класама и функцијама у Пајтон окружењу. Циљ рада је унапређење процеса миграције индустријских апликација у модерније, отворене и лакше одрживе развојне оквире.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	доц. др Миодраг Ђукић	
	Члан:	проф. др Илија Башичевић	Потпис ментора
	Члан, ментор:	проф. др Мирослав Поповић	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Aleksandar Ćetković	
Mentor, **MN**:		Miroslav Popović, PhD	
Title, **TI**:		A tool that assists translating Structured text into Python by using Mistral LLM on AWS Bedrock platform	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2025	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/30/6/8/11/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering and Computer Communications	
Subject/Key words, **S/KW**:		transpiler, PLC controller, Structured Text, Python, AI model	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This project presents the development of a software tool that assists translating programs written in the Structured Text language, commonly used for programming PLC controllers, into Python code using existing Python libraries. Instead of manual mapping, the tool employs an artificial intelligence model to link function blocks and functions from ST libraries to appropriate classes and functions in the Python environment. The goal of the work is to improve the migration process of industrial applications to more modern, open, and maintainable development frameworks.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Miodrag Đukić, PhD	
	Member:	Ilija Bašičević, PhD	Menthor's sign
	Member, Mentor:	Miroslav Popović, PhD	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ИЗРАДУ ЗАВРШНОГ (BACHELOR) РАДА	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	проф. др Иван Каштелан

Студент:	Александар Ћетковић	Број индекса:	РА 24/2021
Област:	Електротехника и рачунарство		
Ментор:	проф. др Мирослав Поповић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ЗАВРШНОГ (BACHELOR) РАДА:

Алат за помоћ у превођењу Структуираног текста у Пајтон коришћењем ЛЛМ Мистрал на платформи АВС Бедрок

ТЕКСТ ЗАДАТКА:

У оквиру овог дипломског рада потребно је урадити следеће:

1. Упознати се са Структуираним текстом за програмирање ПЛЦ контролера, као и са ЛЛМ Мистрал на платформи АВС Бедрок.
2. Израдити пројекат алата за помоћ у превођењу Структуираног текста у Пајтон коришћењем ЛЛМ Мистрал на платформи АВС Бедрок.
3. Имплементирати алат према израђеном пројекту.
4. Валидирати имплементацију на скупу примера програма у Структуираном тексту поређењем резултата оригиналних и преведених програма.
5. Уочити предности и мане имплементираног алата.
6. Написати текст дипломског рада.

Руководилац студијског програма:	Ментор рада:
проф. др Иван Каштелан	проф. др Мирослав Поповић

Примерак за: ☐ - Студента; ☐ - Ментора

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

_____

Кандидат

_____

Захвалност

Велику захвалност дугујем свом ментору, проф. др Мирославу Поповићу, као и техничком ментору Милану Рељину на помоћи и саветима приликом израде овог рада.

Такође посебну захвалност упућујем својој породици и пријатељима, за сву подршку током процеса школовања.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

PLC	- Programmable Logic Controller - програмабилни логички контролери
IEC 61131	- International Electrotechnical Commission standard for programmable controllers - IEC стандард дефинисан за програмабилне контролере
ST	- Structured Text – програмски језик за програмирање PLC контролера
FB	- Function Block - функцијски блок
AI	- Artificial Intelligence - вештачка интелигенција
LLM	- Large Language Model - велики језички модел
API	- Application Programming Interface – програмски интерфејс апликације
AWS	- Amazon Web Services - платформа која нуди рачунарске ресурсе у облаку
JSON	- JavaScript Object Notation - формат за размену података
B&R	- Bernecker & Rainer - произвођач PLC контролера и софтверских алата (Automation Studio)

1. Увод

Убрзани развој индустрије и потреба за ефикасним управљањем производним процесима довели су до масовне примене индустријске аутоматизације, чије језгро чине PLC контролери. Од раних механичких система као што су воденице и сатови, преко релејне логике, па све до данашњих савремених PLC уређаја, аутоматизација је прошла кроз значајне технолошке трансформације. [1]

PLC контролери обезбеђују стабилан и поуздан начин управљања машинама и постројењима, а њихово програмирање је стандардизовано кроз **IEC 61131-3**, који дефинише више програмских језика, међу којима се **Структуирани Текст (ST)** истиче као најзаступљенији текстуални језик. [2] Иако је ST прилагођен инжењерима без великог програмерског знања, његова затворена природа алата доводи до све већег интересовања за модерније програмске језике, попут Python-а, који нуди већу слободу, лакшу интеграцију са савременим библиотекама и активну заједницу корисника.

Поред многобројних изазова у модернизацији постојећих система, један од највећих јесте **миграција наслеђеног (legacy) кода**, који је често власништво компанија и написан у специфичним варијантама ST-а. Ручно преписивање таквог кода је дуготрајно, захтева добро познавање постојеће логике и подложно је грешкама. Додатни проблем представља недоступност имплементација уграђених функција у комерцијалним алатима, што додатно отежава процес миграције.

Савремени трендови у развоју софтвера и индустријских система све више теже отвореним платформама, које омогућавају већу флексибилност, лакше одржавање и бржу интеграцију са другим системима. У том контексту, Python се намеће као идеалан кандидат за замену или проширење постојећих ST решења, захваљујући својој читљивости, великом броју доступних библиотека и могућностима примене у области **вештачке интелигенције** и машинског учења.

У циљу решавања наведених проблема, овај рад представља развој **транспајлера** који користи **AI модел** за помоћ у превођењу ST кода у Python. За разлику од класичних конвертора, који се ослањају на фиксна правила, предложени алат користи интелигентно мапирање функцијских блокова и структура, чиме се омогућава аутоматизовано и контекстуално прецизније превођење. Овим се значајно убрзава процес миграције и смањује потреба за ручним интервенцијама.

Поред тога, транспајлер је пројектован тако да буде проширив и прилагодљив различитим варијантама ST синтаксе, као и специфичним библиотекама које се користе у индустријској пракси. На тај начин, алат не само да служи као мост између два програмска језика, већ и као основа за даљи развој алата за **аутоматизацију миграције индустријског софтвера**.

2. Теоријске основе

Ово поглавље покрива основне теоријске и техничке основе потребне за започињање имплементације рада.

2.1 Транспајлер

Транспајлер (transpiler) је врста компајлера који преводи изворни код написан у једном програмском језику у еквивалентан изворни код другог програмског језика који се налази на истом нивоу апстракције. За разлику од традиционалних компајлера који преводе код високог нивоа у машински код или бајткод (bytecode), транспајлери раде са језицима сличне сложености и производе читљив код који се може даље обрађивати или извршавати. [3]

Транспајлери се често користе за миграцију постојећих система са застарелих на модерне платформе, интеграцију различитих технологија, или када је потребно искористити предности једног језика задржавајући логику написану у другом. Процес превода обично укључује лексичку и синтаксну анализу изворног кода, мапирање конструкција између језика, и генерисање новог кода који очувава оригиналну функционалност, али користи синтаксу и библиотеке циљног језика. [3]

Када развијамо транспајлере, највећи изазови су како да преведемо код са једног језика на други, а да при том сачувамо његову прецизну функцију и брзину рада. Често се дешава да неке фразе из оригиналног језика немају директан превод у циљаном језику, па треба пронаћи адекватно решење. Важно је пажљиво управљати и типовима података и контролним структурама. Посебан изазов у индустријским системима је то што транспајлери морају да обезбеде да код ради поуздано и у реалном времену, без кашњења или непредвидивог понашања.

2.2 Стандард IEC 61131-3 и Структуирани Текст

Поглавље 2.2 написано је на основу материјала из извора [4], који пружа детаљан преглед стандарда IEC 61131-3 и програмског језика Структуирани Текст.

2.2.1 Стандард IEC 61131-3

Стандард IEC 61131-3 је део ширег стандарда IEC 61131 који се односи на програмабилне логичке контролере (PLC) и њихову архитектуру. Овај стандард дефинише два формална програмска језика и три графичка језика који се користе за развој управљачког софтвера у индустријској аутоматизацији. Циљ стандарда је да обезбеди преносивост, читљивост и одрживост кода, као и лакшу интеграцију између различитих PLC платформи и произвођача.
Језици дефинисани у стандардима су:

- Ladder Diagram (LD), графички
- Function Block Diagram (FBD), графички
- Sequential Function Chart (SFC), графички
- Structured Text (ST), формални
- Instruction List (IL), формални

Међу овим, **Структуирани текст** представља најпримитивнији и најфлексибилнији текстуални језик погодан за сложене алгоритме и контролне логике.

2.2.2 Програмски језик Структуирани Текст

Структуирани текст је један од пет програмских језика дефинисаних стандардом IEC 61131-3, који је намењен програмирању програмабилних логичких контролера (PLC). Овај текстуални језик развијен је како би понудио једноставно програмирање, али и могућност изражавања сложених контролних алгоритама.

ST је настао као одговор на потребу за стандардизованим начином писања напредних алгоритама, који превазилазе могућности класичних графичких језика као што су Ladder Diagram и Function Block Diagram.

По својој структури и синтакси, ST подсећа на Паскал, што га чини разумљивим програмерима који долазе из индустрије развоја софтвера. Истовремено, садржи све што је неопходно за рад у системима управљања у реалном времену. Због тога је ST нарочито погодан за имплементацију математичких прорачуна, PID контролера, обраду сигнала и сложене логичке операције.

2.2.3 Синтакса и структура језика

Структуирани текст користи уређену текстуалну синтаксу, инспирисану Паскалом, што омогућава јасно и систематично програмирање. Свака наредба се завршава тачком и зарезом (;), док се блокови кода организују помоћу кључних речи као што су BEGIN и END. Коментари се уносе у облику (* коментар *) за вишелинијске, или // за једнолинијске коментаре.

Програм написан у ST-у састоји се од дефиниција промењљивих и извршног дела кода. Промењљиве се групишу у секције као што су VAR, VAR_INPUT, VAR_OUTPUT, што омогућава прецизну контролу њихове видљивости и употребе.

Језик нуди широк избор **контролних структура**, које омогућавају писање сложених логичких алгоритама:

- IF-THEN-ELSE структура омогућава извршавање кода на основу логичких услова.
- CASE структура служи као ефикасно решење за проверу једне промењљиве у односу на више вредности.
- Петље FOR, WHILE и REPEAT-UNTIL омогућавају различите облике итерације, било унапред познате, било условне.

ST такође подржава велики број **оператора**, укључујући:

- аритметичке (+, -, *, /, MOD) за рачунске операције,
- логичке (AND, OR, XOR, NOT) за рад са логичким вредностима,
- релационе (=, <>, <, >, <=, >=) за поређење вредности,
- као и битске операторе прилагођене специфичностима PLC система.

2.2.4 Типови података

ST језик дефинише велики број типова података, посебно прилагођених потребама индустријских контролних система.

Основни типови укључују:

- логичке типове: BOOL – за представљање true/false вредности,
- целобројне типове: SINT, INT, DINT, LINT, као и њихове unsigned варијанте (USINT, UINT, итд.),
- типове са покретним зарезом: REAL, LREAL – за рад са децималним вредностима,
- временске типове: TIME, DATE, TIME_OF_DAY – за управљање временским подацима,
- текстуалне типове: STRING, WSTRING – за рад са текстом [2].

Сложени типови података омогућавају структурирање информација:

- Низови (arrays) – груписање елемената истог типа,
- Структуре (structures) – комбиновање различитих типова у једну логичку целину,
- Енумерације (enumerations) – дефинисање именованих константи за бољу читљивост и одрживост кода.

2.2.5 Program Organization Units (POU)

Program Organization Units представља организациони концепт у ST-у који омогућава модуларно програмирање, што значи да се сложене контролне логике могу поделити на мање, лакше одрживе и поново употребљиве целине. Сваки POU има јасно дефинисан интерфејс – састављен од улазних (input) и излазних (output) променљивих – и служи за имплементацију одређене функционалности, без мешања са осталим деловима програма.

Постоје три основне врсте POU јединица:

- **Програми** – Представљају главни извршни део кода који се директно позива у оквиру PLC циклуса извршавања. Они имају приступ глобалним променљивим као и улазно-излазном (I/O) систему контролера. Овде се дефинише шта контролер ради.
- **Функције** – То су јединице кода без стања (stateless) које увек враћају једну вредност, на основу улазних параметара. Не могу мењати глобално стање система, што их чини предвидљивим и погодним за математичке и логичке прорачуне.
- **Функцијски блокови (FB)** – За разлику од функција, FB су са стањем (stateful) и задржавају податке током извршавања. Комбинују податке и функционалност, на сличан начин као класе у објектно-оријентисаном програмирању. Захваљујући томе, веома су корисни за моделирање понашања уређаја, процеса или сложенијих алгоритама.

2.2.6 Развојни алати и окружење за Структуирани Текст

За развој програма у Структуираном Тексту постоји низ специјализованих развојних окружења која пружају подршку за програмирање, дебаговање и тестирање кода. Већина ових алата је комерцијализована и једини начин коришћења је издавање бесплатне лиценце на краћи временски период. Неки од најпознатијих алата су:

- B&R Automation Studio, који сам лично користио,
- Siemens TIA Portal,
- Codesys и други.

2.3 Вештачка интелигенција

Вештачка интелигенција (AI) је област рачунарских наука која се бави прављењем система који могу да обављају задатке које обично ради човек, као што су размишљање, учење и доношење одлука.

Када је реч о развоју софтвера, AI нам помаже да аутоматски обављамо сложене задатке - на пример, анализу и генерисање кода, превођење између различитих програмских језика, тумачење документације и слично.

Данас се AI најчешће развија помоћу машинског учења, а посебно дубоког учења, које користи неуронске мреже да препознају сложене обрасце у подацима. Ове технологије су донеле велику промену у областима као што су обрада природног језика, препознавање слика и аутоматско генерисање кода. [5]

2.3.1 Велики језички модели (LLM)

Велики језички модели представљају најнапредније AI системе који умеју да разумеју и генеришу природни језик и програмски код. Ови модели се ослањају на трансформер архитектуру, која користи посебан механизам пажње да би пратила везе између речи или других елемената у тексту, чак и ако су далеко једни од других.

Трансформер архитектура, представљена у раду „Attention Is All You Need“, донела је револуцију јер је уклонила потребу за традиционалним рекурентним или конволуционим слојевима. Механизам сопствене пажње (self-attention) омогућава моделу да истовремено разматра све делове улаза, што значајно побољшава разумевање контекста и везе у тексту.

Примери ових модела су GPT серија, BERT, Mistral модели и многи други. Обучавају се на огромним количинама текста користећи учење без надзора (unsupervised learning). Тренинг обично има две фазе: прва пред-обука (pre-training) када модел учи опште језичке структуре, а затим фино подешавање (fine-tuning) где се модификује за специфичне задатке и области.

Једна од кључних предности савремених LLM-ова је способност да науче нове задатке само на основу неколико примера у самом промпту, без додатне обуке, што се зове учење у контексту (in-context learning). Такође, већи модели показују изненађујуће и нове способности које се појављују како модели расту. Неки од најновијих LLM-ова имају и вишемодалне могућности, што значи да могу да раде са различитим врстама података, као што су текст, слике и звук.

Захваљујући свему овоме, велики језички модели одлично раде у задацима попут генерисања кода, превођења језика, резоновања и решавања сложених проблема. [6]

2.3.2 AWS Bedrock и Mistral Large модел

Bedrock је платформа компаније Amazon Web Services која омогућава приступ водећим моделима вештачке интелигенције кроз јединствен API.

Bedrock користи стандардни API приступ преко HTTP протокола. Python развојни програми могу приступити Bedrock сервису користећи boto3 библиотеку, што омогућава једноставну интеграцију са постојећим системима. API прима захтеве у JSON формату и враћа одговоре foundational модела у структурираном облику.

Један од модела доступан на AWS Bedrock-у је и Mistral Large. Компанија Mistral AI развија широк спектар великих језичких модела, од лагих варијанти као што је Mistral 7B до моћних модела високог капацитета попут Mistral Large. Већина њихових модела је доступна са отвореним тежиштима под лиценцама попут Apache 2.0.

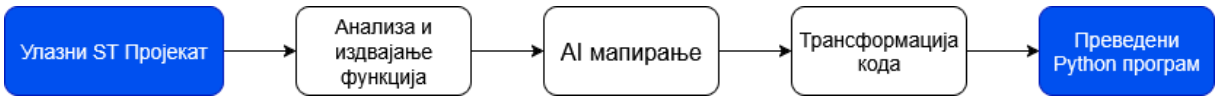
Mistral Large поседује контекстуални прозор од 128.000 токена и подржава десетине језика и више од 80 програмских језика. На MMLU (Massive Multitask Language Understanding) тестирању, Mistral Large постиже тачност од 84.0%, конкуришући водећим моделима попут GPT-4o, Claude 3 Opus и Llama 3 405B. Посебан напор је уложен у унапређење способности резоновања и минимизацију халуцинација, при чему је модел трениран да признаје када нема довољно информација за поуздан одговор, што је посебно битна ствар за овај пројекат.

3. Концепт решења

У овом поглављу биће представљен концепт решења, као и главне идеје и приступи који ће бити коришћени. Поред тога, биће објашњени кључни елементи и структура целокупног система.

3.1 Архитектура пројекта

Систем је организован као секвенцијални ток обраде од три главна модула који раде заједно да реше проблем мапирања библиотечких функција у преведеном Python коду. Сваки модул има јасну одговорност и комуницира са следећим кроз дефинисане интерфејсе.



Слика 3.1 - Архитектура пројекта

3.1.1 Модул 1: Анализа и издвајање функција

Овај модул анализира оригинални ST програм и библиотеке да би идентификовао релевантне функције које треба заменити. На основу добијене листе функција врши се прикупљање и обједињавање пратеће документације.

3.1.2 Модул 2: AI мапирање

Централни модул има улогу посредника између претходног модула и великог језичког модела. Он прикупља све неопходне улазне податке који су резултат рада претходних модула - листу релевантних ST функција, њихову техничку документацију као и доступне Python библиотеке. Поред тога, овај модул формулише прецизне инструкције за велики језички модел, односно упутства о томе како треба да изгледа жељени излаз.

Циљ овог модула је да структуира комуникацију са LLM-ом на такав начин да добијени одговор буде што релевантнији и употребљивији, са минималном потребом за додатном обрадом од стране програмера.

3.1.3 Модул 3: Трансформација кода

Овај модул има задатак да примени резултате које је генерисао велики језички модел на Python код, који је претходно настао превођењем ST програма један-на-један транспајлером и који у том облику још увек садржи библиотечке ST функције. Конкретно, модул врши замену оригиналних ST функцијских позива еквивалентним Python имплементацијама које је предложио LLM током фазе мапирања. Поред саме замене функцијских позива, модул такође додаје потребне import изразе, да би увезао Python библиотеке чије функције се користе. Такође додаје и коментаре изнаде сваке замене коју је направио који објашњавају - на пример, шта је била оригинална ST функција, која је њена Python замена, и шта је основна логика иза ње. Ово олакшава инжењерима и програмерима каснију ручну проверу, дораду или оптимизацију добијеног кода.

4. Програмско решење

У овом поглављу биће детаљно описан начин реализације система, кроз објашњење кључних корака, коришћених техника и међусобне повезаности компоненти. Пројекат је реализован кроз три основна модула, који заједно чине функционалну целину система:

- **functions_extractor.py** - модул за анализу и издвајање библиотечких функција,
- **llm_integration.py** - модул за интеграцију са AI платформом,
- **mapping_applier.py** - модул за трансформацију и замену функција у Python коду.

4.1 Анализа и издвајање библиотечких функција и њихове документације

Систем за анализу и издвајање библиотечких функција је један од кључних делова превода, а имплементиран је у модулу *functions_extractor.py*. Користи комбинацију регуларних израза (regex) за препознавање шаблона у коду и контекстуалну анализу, уз повезивање са релевантним деловима PDF документације.

Процес анализе се одвија кроз четири секвенцијалне фазе:

- **Анализа и издвајање функција из ST програма** - Парсирање кода и идентификација позива библиотечких функција,
- **Препознавање доступних функција из библиотеке** - Екстракција дефиниција из .fun датотеки,
- **Пресек и избор функција за мапирање** - Филтрирање заједничких функција,
- **Издвајање документације за изабране функције** - Проналажење и спајање PDF докумената.

4.1.1 Анализа и издвајање функција из ST програма

Приликом анализе ST програма, систем спроводи детаљну лексичку и синтаксну анализу кода како би идентификовао све библиотечке функције које захтевају мапирање на Python еквиваленте. Ова фаза обухвата комплетну анализу ST кода, која комбинује више техника препознавања образаца, и имплементирана је у функцији *extract_called_functions_from_st_file()*.

Процес почиње уклањањем ST коментара помоћу шаблона регуларних израза који препознаје блокове коментара окружене звездицама и заградама.

Систем користи специјализоване регуларне изразе за препознавање различитих облика позива функција у ST коду. Најпре се користи шаблон за идентификацију класичних функцијских позива са параметрима, који се јављају у облику ИмеФункције(...).

Улазни код	Препознато име функције
`RampController_Execute(setpoint, current, rate)`	`RampController_Execute`
`ArUserHasRole(user_id, role_name)`	`ArUserHasRole`

Табела 4.1 - Пример препознавања функцијских позива

Овај облик позива је карактеристичан за стандардне и кориснички дефинисане функције које се директно позивају у оквиру главног тела ST програма.

За препознавање метода функцијских блокова који се позивају преко инстанци, систем користи додатни шаблон регуларних израза који идентификује приступ члановима у облику Инстанца.Члан := ... или Инстанца.Метод(...).

Улазни код	Препозната инстанца
`signal_integrator.Process(input_signal)`	`signal_integrator`
`Filter.OUT := processed_value`	`Filter`

Табела 4.2 - Пример препознавања инстанци функцијских блокова

Да би се разумело о којим функцијским блоковима је реч, потребно је повезати инстанце са њиховим типовима. То се постиже анализом .var датотеке:

```
VAR
	Ramp : RampController;
	Filter : DeadbandFilter;
	IntegratorFB : SignalIntegrator;
END_VAR
```

Табела 4.3 - Пример .var датотеке

Из овога се гради мапа:

{'Ramp': 'RampController', 'Filter': 'DeadbandFilter', 'IntegratorFB': 'SignalIntegrator'}

Овим кораком систем добија прецизну листу функцијских блокова који се користе у програму и који потенцијално захтевају мапирање на Python библиотеку.

4.1.2 Препознавање доступних функција из библиотеке

Друга фаза процеса фокусира се на анализу библиотечких дефиниција како би се утврдило које библиотечке функције су доступне за коришћење. Функција *extract_defined_functions_from_fun_file()* обрађује .fun датотеке који садрже спецификације функционалних блокова и функција.

.fun датотека у оквиру ST пројекта садржи декларације свих функцијских блокова и функција које су доступне у тој библиотеци. Систем у оквиру процеса превођења користи ову датотеку да би издвојио називе свих функција дефинисаних у библиотеци. То се постиже парсирањем кода и препознавање шаблона FUNCTION и FUNCTION_BLOCK.

```
{REDUND_ERROR} {REDUND_UNREPLICABLE} FUNCTION_BLOCK RampController (*Smoothly ramps a signal to a target value based on rate limits*)
                                                    (*$GROUP=User,$CAT=User,$GROUPICON=User.png,$CATICON=User.png*)
    VAR_INPUT
        TARGET : {REDUND_UNREPLICABLE} REAL;
        CURRENT : {REDUND_UNREPLICABLE} REAL;
        RAMP_UP : {REDUND_UNREPLICABLE} REAL;
        RAMP_DOWN : {REDUND_UNREPLICABLE} REAL;
        ENABLE : {REDUND_UNREPLICABLE} BOOL;
        RESET : {REDUND_UNREPLICABLE} BOOL;
    END_VAR
    VAR_OUTPUT
        OUT : {REDUND_UNREPLICABLE} REAL;
        REACHED : {REDUND_UNREPLICABLE} BOOL;
    END_VAR
    VAR
        PrevOut : {REDUND_UNREPLICABLE} REAL := 0.0;
    END_VAR
END_FUNCTION_BLOCK

{REDUND_ERROR} {REDUND_UNREPLICABLE} FUNCTION_BLOCK DeadbandFilter (*Removes small signal fluctuations around zero or any set point
                                                    useful for noise suppression in sensor readings*)
                                                    (*$GROUP=User,$CAT=User,$GROUPICON=User.png,$CATICON=User.png*)
    VAR_INPUT
        IN : {REDUND_UNREPLICABLE} REAL;
        DEADBAND : {REDUND_UNREPLICABLE} REAL;
        CENTER : {REDUND_UNREPLICABLE} REAL;
    END_VAR
    VAR_OUTPUT
        OUT : {REDUND_UNREPLICABLE} REAL;
        IS_ACTIVE : {REDUND_UNREPLICABLE} BOOL;
    END_VAR
    VAR
        Deviation : {REDUND_UNREPLICABLE} REAL;
    END_VAR
END_FUNCTION_BLOCK
```

Слика 4.1 - Пример .fun датотеке

4.1.3 Пресек и избор функција за мапирање

Трећа фаза представља кључни корак у ком се врши филтрирање и из листе функција се издвајају само оне које су истовремено:

1. **Позване у ST програму** - идентификоване у фази 1
2. **Дефинисане у библиотеци** - пронађене у фази 2

Циљ је да се елиминишу све функције које нису ни коришћене ни доступне за мапирање, како би се систем фокусирао само на релевантне.

Централна функција *compare_function_outputs()* имплементира једноставну али ефикасну логику - издваја заједничке елементе између позваних и дефинисаних функција, односно враћа само оне које се налазе у оба скупа.

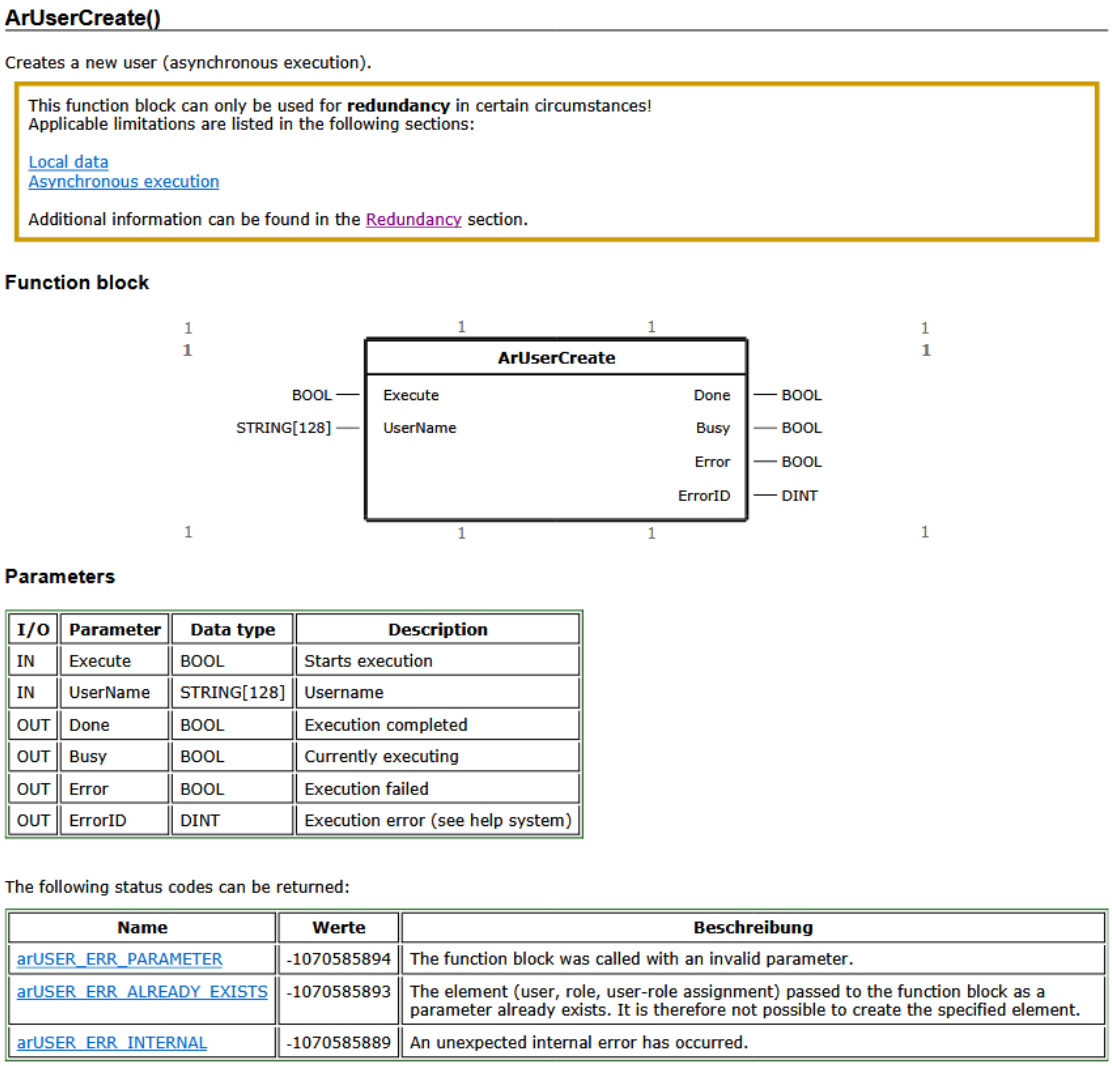
```
def compare_function_outputs(called_functions, defined_functions):
   return sorted(set(called_functions) & set(defined_functions))
```

Табела 4.4 - Функција за пресек коришћених функција

Резултат извршавања ове функције је коначна листа функција које се и заиста позивају у ST програму и истовремено постоје у библиотеци. Та листа се затим чува у датотеци *common_functions.txt* и касније користи као улаз за припрему документационог контекста и слање AI моделу.

4.1.4 Издвајање документације за изабране функције

Четврта и финална фаза овог модула аутоматски обједињава PDF документацију за све функције идентификоване у претходној фази. Систем користи функцију *prepare_and_merge_documentation()* која координира поступак издвајања релевантних PDF датотека и њиховог спајања у један излазни документ. Пре овог корака неопходно је преузети оригиналну документацију са сајта произвођача библиотека, како би систем имао приступ свим потребним датотекама.



ArUserCreate()

Creates a new user (asynchronous execution).

This function block can only be used for **redundancy** in certain circumstances!
Applicable limitations are listed in the following sections:

Local data
Asynchronous execution

Additional information can be found in the Redundancy section.

Function block

Parameters

I/O	Parameter	Data type	Description
IN	Execute	BOOL	Starts execution
IN	UserName	STRING[128]	Username
OUT	Done	BOOL	Execution completed
OUT	Busy	BOOL	Currently executing
OUT	Error	BOOL	Execution failed
OUT	ErrorID	DINT	Execution error (see help system)

The following status codes can be returned:

Name	Werte	Beschreibung
arUSER_ERR_PARAMETER	-1070585894	The function block was called with an invalid parameter.
arUSER_ERR_ALREADY_EXISTS	-1070585893	The element (user, role, user-role assignment) passed to the function block as a parameter already exists. It is therefore not possible to create the specified element.
arUSER_ERR_INTERNAL	-1070585889	An unexpected internal error has occurred.

Слика 4.2 - Пример документације

Документација је организована у стандардизовану структуру директоријума:

```
Dokumentacija/
├── RampController/          # Прва библиотека
│   ├── RampController.pdf     # Документ за главну функцију
│   └── DeadbandFilter.pdf     # Документ за филтер функцију
├── SignalLib/               # Друга библиотека
│   ├── SignalIntegrator.pdf
│   ├── SignalLimiter.pdf
│   └── ToggleDebounce.pdf
└── ArUser/                  # Трећа библиотека
    ├── ArUserHasRole.pdf
    └── ArUserCreate.pdf
```

Табела 4.5 - Организација документације

Током ове фазе, систем претражује све поддиректоријуме, идентификује датотеке чија имена одговарају функцијама из листе *common_functions.txt* и додаје их у заједнички PDF. На овај начин се добија обједињен документ који представља базу знања за све функције које су потребне.

4.2 Интеграција са AI платформом

Након што су издвојене све релевантне функције из ST програма, а затим и обједињена њихова документација, систем прелази у фазу семантичког мапирања. Овај модул представља срце целог пројекта, јер комбинује све дотад прикупљене податке (коришћене функције, документацију, Python библиотеку) и шаље их великом језичком моделу путем AWS Bedrock API-ја.

4.2.1 Структура и припрема улаза

Главна скрипта која садржи целу логику ове фазе је *llm_integration.py.* Она учитава следеће улазне компоненте:

- Списак коришћених библиотечких функција које су издвојене током прошле фазе у датотеку *common_functions.txt,*
- Обједињена PDF документација за сваку од тих функција (*merged_output.pdf),*
- Python библиотека.

Да би ово било могуће, скрипта садржи више наменских функција:

- *load_file(file_path)* - учитавање садржаја датотека (PDF преко fitz библиотеке или обичних текстуалних датотека).
- *call_bedrock_api(prompt: str)* - успоставља сесију са AWS Bedrock API-јем, шаље упит (prompt) и враћа одговор модела у JSON формату AWS Bedrock и избор модела.
- *__main__* блок - оркестрира цео процес: учитава улазне компоненте, формулише упит са строгим правилима и прослеђује га великом језичком моделу.

4.2.2 AWS Bedrock и избор модела

За реализацију комуникације са великим језичким моделом изабран је **AWS Bedrock** - сервис који омогућава лак приступ већем броју модела кроз једноставну API спрегу. Једна од највећих предности AWS-овог Bedrocka су уграђени безбедности механизми руковања документацијом и кодом, што је значајно у случајевима где се ради са потенцијално осетљивим или власничким подацима.

У конкретној имплементацији, изабран је модел **Mistral Large**, који се показао најбоље показао у погледу цене, квалитета и ефикасности у задацима који захтевају анализу документације, препознавање семантичких образаца и доношења мапирајућих одлука.

За интеграцију са AWS Bedrock-ом коришћена је званична Python библиотека **boto3**, која служи за програмски приступ свим AWS услугама, па самим тим и Bedrock-у. Предност библиотеке је у томе што сакрива сложеност директне HTTP комуникације са Bedrock API-јем и омогућава конфигурисање параметара, формат улаза и излаза.

4.2.3 Концепт упита-а и изазови формулације

Прецизно и правилно формулисан упит (prompt) је један од најкритичнијих елемената целог система. Ако LLM не добије добро дефинисане инструкције, сам излаз из модела може да буде у потпуности бескроистан.

Највећи изазови при формулисању инструкција је баланс између:

- **Довољне количине контекста** – модел мора да има приступ опису функције, њеној улози у ST програму и вези са библиотеком, како би могао да изведе тачно мапирање. У супротном, постоји ризик од измишљања (*hallucination)* нових функција.
- **Јасних ограничења** – у упиту је неопходно строго дефинисати оквир, тј. које Python библиотеке се смеју користити, које функције су дозвољене, а које треба избећи. Ово спречава модел да генерише неконстантне или небезбедне предлоге.
- **Формалне структуре излаза** – поред садржаја, подједнако је важно да одговор буде унапред дефинисаног формата, јер једино тако каснији модули могу аутоматски да га обраде без додатне ручне интервенције.

Прве верзије упита су давале непоуздане резултате. Модел је често предлагао непостојеће Python функције, користио неодговарајуће библиотеке, или генерисао излазе који нису имали јасну структуру, што је онемогућавало њихову аутоматску обраду. Тиме је постало очигледно да је ижењерство упита (*prompt engineering)* сложен процес који захтева итеративно унапређивање.

4.2.4 Формат излаза и чување резултата

Излаз из ове фазе је листа мапирања у облику „ST_Function → Python_Function“. Резултат се уписује у текстуалну датотеку, што омогућава лаку размену података између модула и независност од конкретне имплементације. На тај начин је могуће поновити поступак на различитим програмима или библиотекама без измене централне логике. Такође, чување резултата у текстуалном формату омогућава и једноставну проверу исправности од стране корисника, као и верзионисање током развоја.

```
DeadbandFilter → Filter.process
RampController → Controller.update, Controller.reset
```

Табела 4.6 - Пример излаза из ЛЛМ-а

Овај пример илуструје два кључна аспекта:

- Једна ST функција може бити директно мапирана на једну Python функцију
- Једна ST функција може бити разложена на више Python функција које заједно реализују исту логику

На овај начин систем није ограничен само на тривијалне случајеве један-према-један, већ подржава и сложеније сценарије где је потребно разложити или реконфигурисати функционалност. Управо ова флексибилност представља важну предност у односу на статичке приступе трансформацији кода.

4.3 Трансформација и замена функција у Python коду

Ова фаза примењује мапирања из претходног корака на генерисани Python код и представља финалну трансформацију ST кода у Python. Процес се одвија кроз неколико фаза које омогућавају прецизну и сигурну замену без нарушавања структуре кода.

Имплементација је реализована у скрипти *mapping_applier.py*, која систематски спроводи добијена правила мапирања и ажурира излазни код у складу са дефинисаним трансформацијама.

4.3.1 Парсирање мапирања

Овај део је имплементиран у функцији *parse_function_mapping()*. Као улаз користи се излазни резултат претходне фазе у коме су дефинисана мапирања у облику „ST_Function →

Python_Function“. Да би се ова мапирања правилно интерпретирала, примењује се шаблон регуларних израза који омогућава препознавање различитих варијација записа:

- игноришу се почетни набрајачки знаци (-, *, бројеви са тачком),
- дозвољено је да се функција појави у наводницима или са „*“/„`“,
- раздвајач може бити „→“, „–“, „-“, „=“ (1–3 симбола),
- десна страна се узима као целина (може садржати више Python симбола, раздвојених зарезима).

Улазни сет мапирања	Парсирана CT функција	Парсирана Пајтон функција
`DeadbandFilter → Filter.process`	`DeadbandFilter`	`Filter.process`
`* DeadbandFilter - Filter.process, Filter.update_output`	`DeadbandFilter`	`Filter.process, Filter.update_output`
`` `DeadbandFilter` = `Filter.process` ``	`DeadbandFilter`	`Filter.process`

Табела 4.7 - Примери улаза и резултата парсирања

Резултат обраде је речник у коме се имена функцијских блокова из ST библиотека повезују са њиховим Python еквивалентима.

4.3.2 Замена инстанцирања функцијских блокова

У следећем кораку систем препознаје линије кода у којима се инстанцирају стари ST блокови у облику променљива = стари функцијски блок. Оне се замењују новим класама на основу изграђене мапе. Овај поступак је реализован у функцији *replace_function_blocks()*.

Пример трансформације за следећу мапу функција:

```
SignalIntegrator → Intergrator.accumulate, Intergrator.update_output
SignalLimiter → SignalController.clamp_signal,SignalController.apply_hysteresis
ToggleDebounce → Toggler.detect_edge
```

Пре	После

`LimiterFB = SignalLimiter()`
`DebounceFB = ToggleDebounce()`
`IntegratorFB = SignalIntegrator()`
	`LimiterFB = SignalController()`
	`DebounceFB = Toggler()`
	`IntegratorFB = Intergrator()`

Табела 4.8 - Пример трансформације мапе функција

4.3.3 Изградња мапе променљивих

Да би се касније омогућило додавање коментара ради чувања трага, систем изграђује мапу која повезује имена променљивих са оригиналним ST блоковима. На пример, ако у Python коду постоји инстанца ramp_controller, потребно је знати да је она настала од RampController блока. На овај начин омогућено је чување трага трансформације, што је посебно важно за касније развој и дебаговање. Управо то реализује функција *find_variable_to_class_map()*.

4.3.4 Убацивање коментара и import наредби

Да би генерисани Python код био извршив, неопходно је обезбедити доступност нових класа. Уколико у коду већ не постоји одговарајући import, систем га додаје након последње постојеће import наредбе. За то је задужена функција *insert_library_import()*, која води рачуна да се у коду не појаве дуплирани увози.

Изнад линија у којима се позивају методе функцијских блокова систем додаје коментаре који указују на изворно мапирање. На тај начин обезбеђује се јасна веза између трансформисаног кода и оригиналне ST имплементације, што је важно за праћење и дебаговање. Овај механизам реализује функција *insert_function_comments()*, која користи мапу променљивих да би знала на

који ST блок се свака Python инстанца односи. Коментари омогућавају програмерима да одмах уоче који део кода је измењен ради интеграције функцијских блокова, као и да у случају потребе лако изврше додатне измене или прилагођавања.

5. Резултати

У овом поглављу приказани су резултати провере рада развијеног система за мапирање ST библиотечких функција на Python функције. Циљ је био да се утврди колико систем верно препознаје и замењује функцијске блокове. Резултати су илустровани кроз неколико примера који показују како систем ради у различитим ситуацијама, као и кроз поређење добијеног Python кода са оригиналним ST записима. На тај начин може се сагледати колико је решење поуздано и примењиво у пракси.

5.1 Ограничења тестирања

Иако је систем показао успешност на припремљеним примерима, важно је истаћи да обим тестирања није обухватио праве индустријске пројекте. Главни разлози за ово ограничење су:

- **Недоступност хардвера и извршног окружења** - ST програме није било могуће покренути директно на B&R контролерима, већ искључиво у симулатору, па самим тим није било могуће упоредити резултате са Python имплементацијом у реалном систему.
- **Недостатак великих индустријских библиотека** - коришћене су само доступне B&R библиотеке, док су сложени пакети произвођача који се користе у индустријској пракси остали недоступни.
- **Ограничени ресурси** - тестирање је спроведено у симулираном окружењу са ограниченом количином функцијских блокова.

Због ових фактора, резултати треба да се тумаче као **валидна демонстрација концепта** и доказ исправности система у контролисаним условима, а не као свеобухватна евалуација у индустријској употреби.

5.2 Тестни случајеви

Систем је тестиран на три B&R Automation Studio пројекта, изабрана тако да покривају различите домене индустријске аутоматизације (сигнална обрада, управљање процесима, безбедност и корисничке улоге). Овакав избор омогућио је процену система у реалним и разноврсним сценаријима.

5.2.1 RampController пројекат

Овај пројекат обухвата контролу рампи и филтрирање сигнала са зоном неосетљивости (deadband). Резултати показују да је систем успешно препознао и мапирао функцијске блокове RampController и DeadbandFilter, те их правилно пресликао на Python еквиваленте.

```
PROGRAM _CYCLIC
    (** --- RAMP CONTROLLER LOGIC --- **)
    Ramp(TARGET := UserSetpoint, CURRENT := CurrentValue, RAMP_UP := RampRateUp,
        RAMP_DOWN := RampRateDown, ENABLE := RampEnable, RESET := RampReset);
    SmoothSetpoint := Ramp.OUT;
    AtTarget := Ramp.REACHED;

    (** --- DEADBAND FILTER LOGIC --- **)
    Filter(IN := SensorRaw, DEADBAND := Deadband, CENTER := SmoothSetpoint);
    SensorFiltered := Filter.OUT;
    SensorActive := Filter.IS_ACTIVE;

END_PROGRAM
```

Слика 5.1 - Улазни RampController програм

```
Python > outputs > Ramp_output.txt
1   ```
2   DeadbandFilter → Filter.process
3   RampController → Controller.update, Controller.reset
4   ```
```

Слика 5.2 - Излаз из LLM-а за RampController

```
def _CYCLIC(self):
    #Ramp → Controller.update, Controller.reset
    self.Ramp(TARGET = self.UserSetpoint, CURRENT = self.CurrentValue, RAMP_UP = self.RampRateUp,
              RAMP_DOWN = self.RampRateDown, ENABLE = self.RampEnable, RESET = self.RampReset)
    self.SmoothSetpoint = self.Ramp.OUT
    self.AtTarget = self.Ramp.REACHED
    #Filter → Filter.process
    self.Filter(IN = self.SensorRaw, DEADBAND = self.Deadband, CENTER = self.SmoothSetpoint)
    self.SensorFiltered = self.Filter.OUT
    self.SensorActive = self.Filter.IS_ACTIVE
    return
```

Слика 5.3 - Python Ramp програм након имплементирања мапираних функција

5.2.2 SignalProcessing пројекат

У другом тесту испитивана је сложенија логика обраде сигнала која укључује интеграцију, ограничавање и смањење одскока (debounce). Резултати мапирања показују да је систем успешно поделио једну ST функцију на више Python метода, као и да је правилно решио сложеније блокове.

```
PROGRAM _CYCLIC

    CycleCounter := CycleCounter + 1;

    IF CycleCounter = 201 THEN
        CycleCounter := 0;
    END_IF;

    IF CycleCounter <= 100 THEN
        ButtonInput := TRUE;
    ELSE
        IF CycleCounter >= 101 THEN
            ButtonInput := FALSE;
        END_IF;
    END_IF;

    LimiterFB(IN := AnalogRaw, MIN_LIMIT := MinLimit, MAX_LIMIT := MaxLimit, HYSTERESIS := 0.2);
    LimiterFB();
    LimitedSignal := LimiterFB.OUT;

    DebounceFB(TRIG := ButtonInput, DEBOUNCE_CYCLES := deb_cyc);
    DebounceFB();

    IntegratorEnabled := DebounceFB.VALID_EDGE;

    IntegratorFB(IN := LimitedSignal, DT_STEP := DtStep, ENABLE := IntegratorEnabled,
                 RESET := ResetFlag, MAX_ACCUM := MaxAccum, DELTA_LIMIT := DeltaLimit);
    IntegratorFB();

    AccumulatedResult := IntegratorFB.OUT;
END_PROGRAM
```

Слика 5.4 - Улазни SignalProcessing програм

Python > outputs > SignalProcess_output.txt

```
1  ```
2  SignalIntegrator → Intergrator.accumulate, Intergrator.update_output
3  SignalLimiter → SignalController.clamp_signal, SignalController.apply_hysteresis
4  ToggleDebounce → Toggler.detect_edge
5  ```
```

Слика 5.5 - Излаз из LLM-а за SignalProcessing

```
def _CYCLIC(self):
    #LimiterFB → SignalController.clamp_signal, SignalController.apply_hysteresis
    self.LimiterFB(IN = self.AnalogRaw, MIN_LIMIT = -10.0, MAX_LIMIT = 10.0, HYSTERESIS = 0.2)
    self.LimiterFB
    self.LimitedSignal = self.LimiterFB.OUT
    #DebounceFB → Toggler.detect_edge
    self.DebounceFB(TRIG = self.ButtonInput, DEBOUNCE_CYCLES = 20)
    self.DebounceFB
    if self.DebounceFB.VALID_EDGE:
        self.IntegratorEnabled =  not self.IntegratorEnabled
    #IntegratorFB → Intergrator.accumulate, Intergrator.update_output
    self.IntegratorFB(IN = self.LimitedSignal, DT_STEP = 0.01, ENABLE = self.IntegratorEnabled,
                      RESET = False, MAX_ACCUM = 100.0, DELTA_LIMIT = 0.5)
    self.IntegratorFB
    self.AccumulatedResult = self.IntegratorFB.OUT
    return
```

Слика 5.6 - Python Signal програм након имплементирања мапираних функција

5.2.3 ArUser пројекат

У трећем случају тестиране су функције за управљање корисницима. Ово је посебно значајно јер обухвата безбедносне механизме (аутентификацију и ауторизацију). Иако је систем без грешке мапирао све три тестиране функције (ArUserHasRole, ArUserCreate, ArUserAuthenticatePassword) на одговарајуће Python функције, није било могуће интегрисати их у комплетан извршни код.

Због дужине главног програма, на Слици 10 дат је издвојени исечак који приказује типичан ток: позив функцијског блока и обраду исхода преко поља .Done и .Error. Остали кораци (подешавање лозинке, креирање и додела улога) имплементирани су идентичним шаблоном.

```
9:
    authPassword.UserName := 'demoUser7';
    authPassword.Password := 'demoPass123';
    authPassword.Execute := TRUE;
    step := 10;

10:
    authPassword();
    IF authPassword.Error THEN
        authPassword.Execute := FALSE;
        step := 11;
    ELSIF authPassword.Done THEN
        authPassword.Execute := FALSE;
        step := 12;
    END_IF;
```

Слика 5.7 - Део улазног ArUser програм

Python > outputs > UserRole_output.txt

```
1  - `ArUserAssignRole` → `UserAssignRole.call`
2  - `ArUserAuthenticatePassword` → `UserAuthenticatePassword.call`
3  - `ArUserCreate` → `UserCreate.call`
4  - `ArUserCreateRole` → `UserCreateRole.call`
5  - `ArUserSetPassword` → `UserSetPassword.call`
```

Слика 5.8 - Излаз из LLM-а за ArUser

Разлог за ово ограничење лежи у чињеници да главни програм (main) користи специфичне B&R Automation Runtime конструкције које немају директан еквивалент у Python окружењу. Конкретно, програм се ослања на механизме за распоређивање задатака (task scheduling) који су уграђени у B&R систем. Поред тога, ArUser функције користе интерне B&R системске позиве (system calls) за приступ модулима хардверске заштите (hardware security) који су специфични за B&R платформу.

Овај случај илуструје фундаментално ограничење приступа превођења по принципу један-на-један - док се појединачне функције могу успешно мапирати, системске зависности и извршно окружење могу спречити потпуну миграцију сложенијих апликација које се ослањају на специфичане за произвођача инфраструктурне компоненте.

5.3 Анализа перформанси система

Перформансе система нису систематски тестиране на већим пројектима, већ је развој и валидација рађена на мањим примерима. Због тога приказани резултати не одражавају реално понашање система у продукционом окружењу и не могу се користити као мерило скалабилности. Време одзива, оптерећеност сервера и количина података који се прослеђују моделу у пракси могу значајно варирати, али ови параметри нису били предмет мерења у оквиру овог рада.

Фокус истраживања био је пре свега на функционалној тачности и коректности мапирања функција, док је процена перформанси остављена за будући рад. За реалну евалуацију потребно је спровести тестирања на већим пројектима, уз јасно дефинисане метрике као што су просечно и максимално време одзива и потрошња ресурса.

5.4 Резиме тестирања

Спроведено тестирање на три различита B&R Automation Studio пројекта пружило је почетне увиде у функционисање развијеног система за мапирање ST функција на Python еквиваленте. Међутим, ограничен број тестних случајева представља значајну препреку за доношење поузданих закључака о стварној ефикасности система.

Овако мала база података не омогућава исправно преношење резултата на већу групу индустријских PLC програма, који се знатно разликују по сложености, грађи и врстама функцијских блокова. Недостаје представљеност узорка који би обухватио различите индустријске гране, врсте контролера и специфична решења произвођача.

Иако су почетни резултати охрабрујући, потребно је проширење тестне базе са библиотекама из разних индустријских окружења пре него што се могу донети поуздани закључци о применљивости алата. Такође је неопходно спровести упоредну анализу са другим LLM моделима и традиционалним миграционим приступима како би се објективно проценила вредност предложеног решења.

6. Закључак

Резултати овог рада показују да је могуће осмислити и имплементирати софтверски алат који на интелигентан начин омогућава помоћ у превођењу програма написаних у језику Структуирани текст у Python код, уз помоћ великих језичких модела. Развијено решење демонстрира да је могуће убрзати процес миграције индустријских апликација, смањити ризик од грешака при ручном преписивању и обезбедити основу за будући развој аутоматизованих алата за модернизацију софтвера.

Провера функционисања система извршена је у оквиру расположивих ресурса и показује задовољавајуће резултате у одређеним границама. Иако није било могуће спровести тестирање на сложенијим реалним примерима какви се срећу у индустријској пракси, алат демонстрира способност успешне обраде значајног дела пројеката. У ситуацијама где аутоматско мапирање није дало резултате или је резултат непотпун, одговорност за даље поступање остаје на инжењерима.

У том контексту, коришћење великог модела као што је Mistral Large може се посматрати и као полазна тачка за будућа истраживања у правцу развоја мањих, специјализованих модела оптимизованих за овакве задатке. Такви модели би могли бити једноставнији за примену, захтевати мање ресурса и пружати већу поузданост у специфичним индустријским сценаријима.

Закључно, овај рад представља један од првих корака ка интеграцији савремених техника вештачке интелигенције у процес миграције наслеђених PLC апликација, са циљем да се оне приближе софтверским решењима која су модерна, отворена и једноставна за одржавање.

7. Литература

[1] “Programmable Logic Controller”, [Online].
Available: https://en.wikipedia.org/wiki/Programmable_logic_controller

[2] “Structured Text”, [Online].
Available: https://en.wikipedia.org/wiki/Structured_text

[3] “Transpiler”, [Online].
Available: https://devopedia.org/transpiler

[4] “Structured Text Programming”, PDHOnline Course E334, [Online].
Available: https://pdhonline.com/courses/e334/e334content.pdf

[5] “Artificial Intelligence”, [Online].
Available: https://en.wikipedia.org/wiki/Artificial_intelligence

[6] “AI, Large Language Models and University Education”, [Online].
Available: https://www.i-aida.org/wp-content/uploads/2023/09/AI-Large-Language-Models-and-Un-Educat-v1.1.pdf