HMS

analytical software

HMS analytical software

From SAS to R/Python

Practical Insights into Automated Code Translation

Dr. Robert Bauer
robert.bauer@analytical-software.de
Senior Data Scientist
HMS Analytical Software GmbH

PHUSE SDE Frankfurt 13.03.2025

Introduction

What we discuss in this talk

- We developed an Expert Tool that **automates code translation from SAS to R/Python** in a reliable and robust way based on Generative AI and sophisticated templating
- Our approach gives us a **high degree of control over the translation output** which is crucial for clinical projects
- The approach significantly **accelerates our T&M projects** through faster completion and reduced costs: We used it in several SAS -> Python projects (insurance and finance sector) with great success and **substantial savings in terms of time and material**
- We recently started our **first SAS -> R migration project** for a large pharmaceutical company

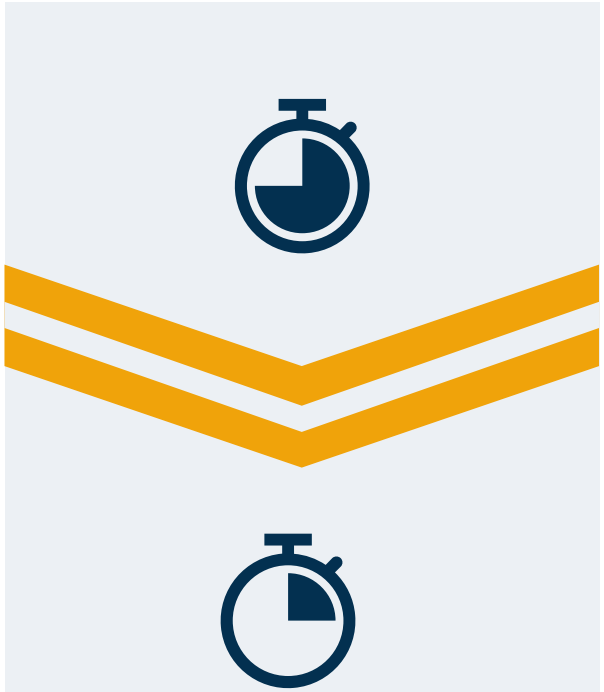
02 | Motivation

Reasons for Transitioning to R/Python

- Utilize features and libraries from another (programming) ecosystem
- Benefit from open source flexibility
- Access to bigger/other talent pools
- Comply with global IT strategy
- Economic Reasons
- …

Why Automatic Code Translation?

- Transition to other programming paradigm is a **big investment** in terms of effort and time
- Especially if you need to migrate **large amounts of code**
- Automatic code translation is a tool to make such migrations viable: we see **savings of at least 50% with Automation**
- Produces **optimized and consistent code** in terms of readability, quality, documentation, …



Goal: Fast, structured and scalable migration projects

Unstructured GenAI Code Translation is not Enough

```
= session.load_df("class", library=

# Apply transformations
df['WEIGHT'] = df.apply(
    lambda row: row['WEIGHT'] * 1.2 if
)
df['WEIGHT'] = df.apply(
    lambda row: row['WEIGHT'] * 0.99 i
)

# Create new columns based on the loop
samples = []
short_names = []
outputs = []

for i in range(1, 4):
    sample = df['HEIGHT'] * i / 3
    short_name = df['NAME'].str.upper(
    samples.append(sample)
    short_names.append(short_name)
    outputs.append(sample > 35)

# Add new columns to DataFrame
df['SAMPLE'] = pd.concat(samples, axis
df['SHORTNAME'] = pd.concat(short_name
df['OUTPUT'] = pd.concat(outputs, axis

# Filter rows where OUTPUT is True
df_output = df[df['OUTPUT']]
```

```
suming the DataFrame is already loaded into `df`
f = pd.read_csv('path_to_your_csv_file.csv')

# Apply the transformations
def transform_row(row):
    if row['sex'] == 'M' and row['name'] <= 'J':
        row['weight'] = row['weight'] * 1.2
    elif row['name'] in ['Jane', 'Janet']:
        row['weight'] = row['weight'] * 0.99

    for i in range(1, 4):
        sample = i * row['height'] / 3
        short_name = row['name'][:3].upper()
        if sample > 35:
            # Assuming you want to collect these rows
            output_row = row.copy()
            output_row['sample'] = sample
            output_row['shortName'] = short_name
            yield output_row

# Create an empty DataFrame to store the output
output_df = pd.DataFrame()

# Iterate over each row and apply the transformation
for _, row in df.iterrows():
    for transformed_row in transform_row(row):
        output_df = output_df.append(transformed_row, ignore_index=True)

# Save the output DataFrame to a new CSV file
output_df.to_csv('path_to_output_csv_file.csv', index=False)
```

```
df = session.load_df("class", library="work")

# Apply the conditions
df.loc[(df['SEX'] == 'M') & (df['NAME'] <= 'J'), 'WEIGHT'] *= 1.2
df.loc[df['NAME'].isin(['Jane', 'Janet']), 'WEIGHT'] *= 0.99

# Create a new DataFrame to store the results of the loop
results = []

# Loop equivalent to Do i=1 To 3;
for i in range(1, 4):
    df['SAMPLE'] = i * df['HEIGHT'] / 3
    df['SHORTNAME'] = df['NAME'].str.upper().str[:3]
    results.append(df[df['SAMPLE'] > 35])

# Concatenate all results
final_df = pd.concat(results)
```

The same input code translated with slightly different prompts using GPT-4

HMS analytical software

Reliable and High Quality Code is the Primary Challenge

Translating snippets with LLMs from one language to another is „easy“

Works quite well without too much effort

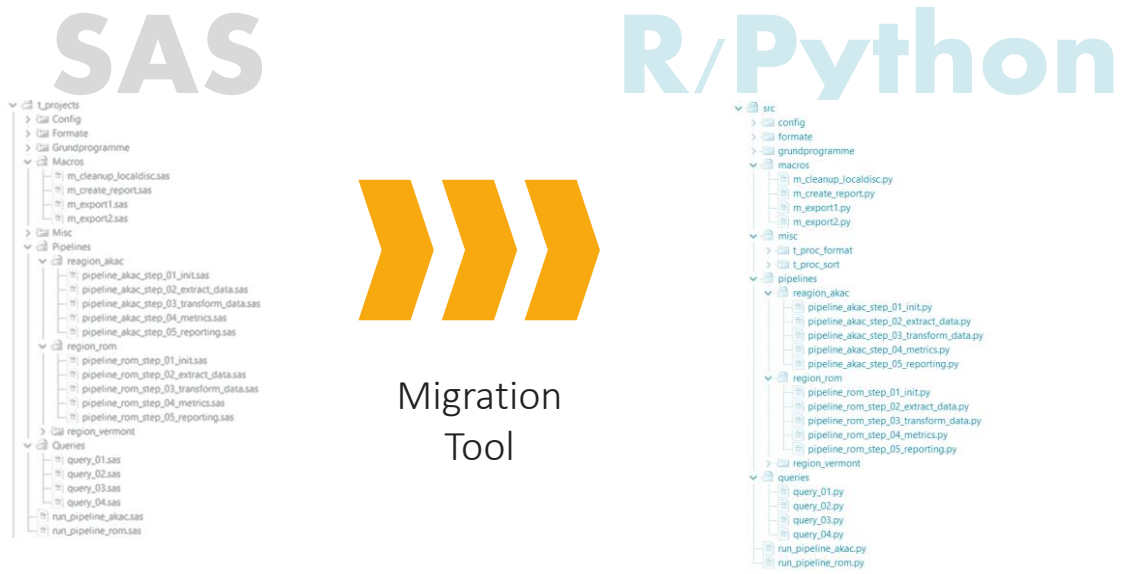
Getting high quality code in a **reliable** matter?

Not so easy but possible!

03 | Our Approach

Approach in a Nutshell

- We build an “**Expert Tool**” to automate code translation to accelerate our T&M projects
- Highly flexible due to code templating
- Can process code **snippets** individually or complete projects via **batch mode**
- However: Not supposed to handle full projects in “one click and done”
- Uses GenAI with care → if we can do something better/faster without GenAI we prefer to do that!



Approach in a Nutshell

Steps 3 and 4 depend on target system architecture!

Step 1

Parser Creates **AST (Abstract Syntax Tree)**

File: **fb01629e604ff75a4f9930940bf0a316**
snippets/2/fb01629e604ff75a4f9930940bf0a316 (id=886)

- Program
  - DatastepV2 DATA
    - StatementSet
    - LoopIf
      - IfConditionV2 (sex = M)
        - StatementOutput
      - IfConditionV2 OTHERWISE
        - StatementOutput

Step 2

Meta Model Analysis for each AST Node

```
MetaModelConditional(
    condition=' (  sex = VAR_STRING_PLCHLDR_000000  ) ',
    clause=MMClauseBinary(
        mm_type='MMClauseBinary',
        operator='eq',
        c1=MMVariable(
            mm_type='MMVariable',
            name='sex',
            variable_type='CNAME',
        ),
        c2=MMVariable(
            mm_type='MMVariable',
            name='VAR_STRING_PLCHLDR_000000',
            variable_type='CNAME',
        ),
    ),
)
```

Step 3

Code Restructuring based on target architecture

File: **fb01629e604ff75a4f9930940bf0a316**
snippets/2/fb01629e604ff75a4f9930940bf0a316 (id=886)

- Program
  - DatastepV2 DATA — OK
    - VNLoadDataset — OK
    - VNPrepareOutput — OK
    - VNPrepareOutput — OK
    - StatementSet
    - LoopIf — OK
      - IfConditionV2 (sex = M) — OK
        - StatementOutput — OK
      - IfConditionV2 OTHERWISE — OK
        - StatementOutput — OK
    - VNSaveDataset — OK
    - VNSaveDataset — OK

Step 4

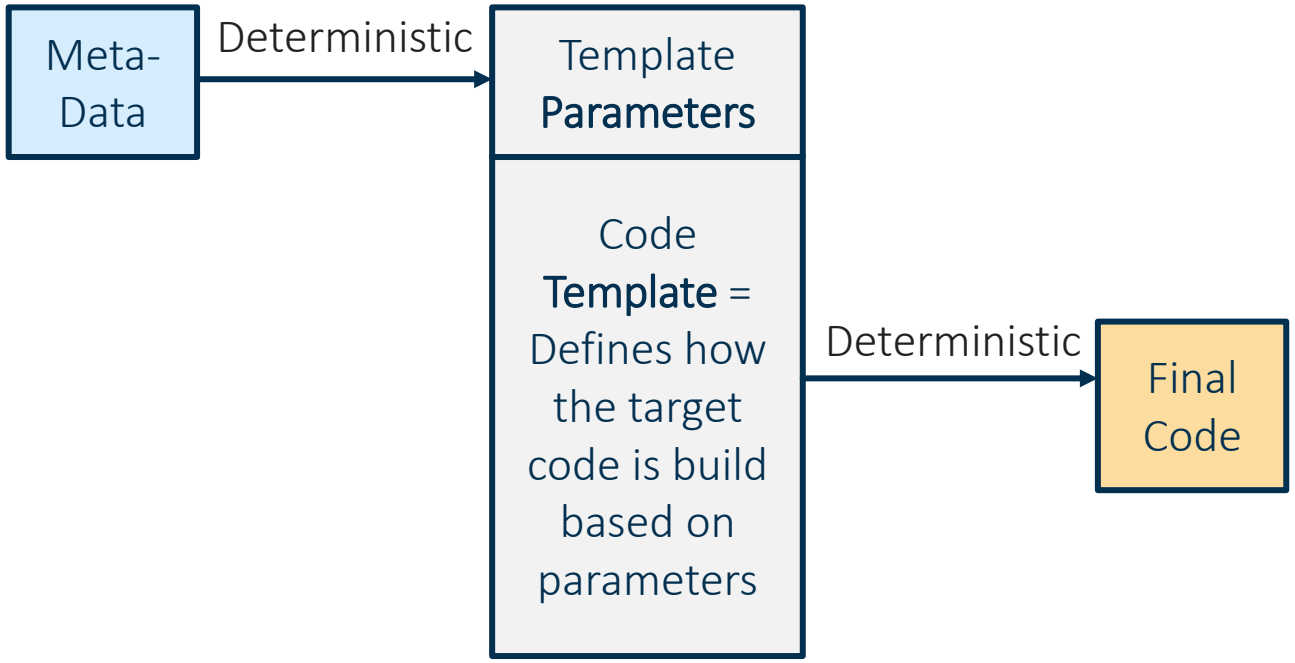
Code Processor Development and actual **code translation**

```
# %%
# ================
# Erstellung von >>Work.Class1<<
# ================
working_df = session.load_df("class", library="work")
df_class1 = working_df.head(0).copy() # used for explicit output
df_class2 = working_df.head(0).copy() # used for explicit output

# %%
session.log_debug("""
> ====================
> Transformation:
> ====================
> [ 1  ] if ( sex = 'M' ) then
> [ 2  ] else
> ====================
""")

# Transformation 1
def _F1_():
    return working_df['SEX'] == 'M'
_C1_ = _F1_()
session.log_debug(f"_C1_:  {len(working_df[_C1_])}/{len(working_df)}")
df_class1 = pd.concat([df_class1, working_df[_C1_]], ignore_index=True)

# Transformation 2
_C2_ = ~_F1_()
session.log_debug(f"_C2_:  {len(working_df[_C2_])}/{len(working_df)}")
df_class2 = pd.concat([df_class2, working_df[_C2_]], ignore_index=True)
session.save_df(df_class1, "class1", library="work")
session.save_df(df_class2, "class2", library="work")
```

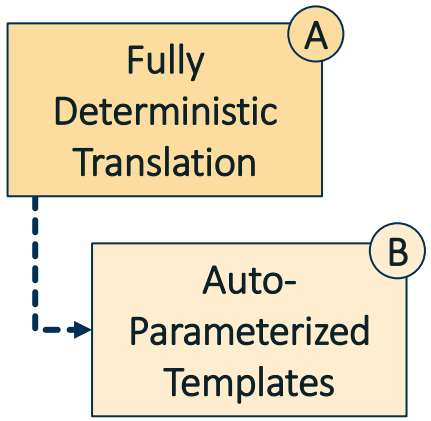
We use GenAI and/or deterministic processes for transition between the steps based on goal and complexity

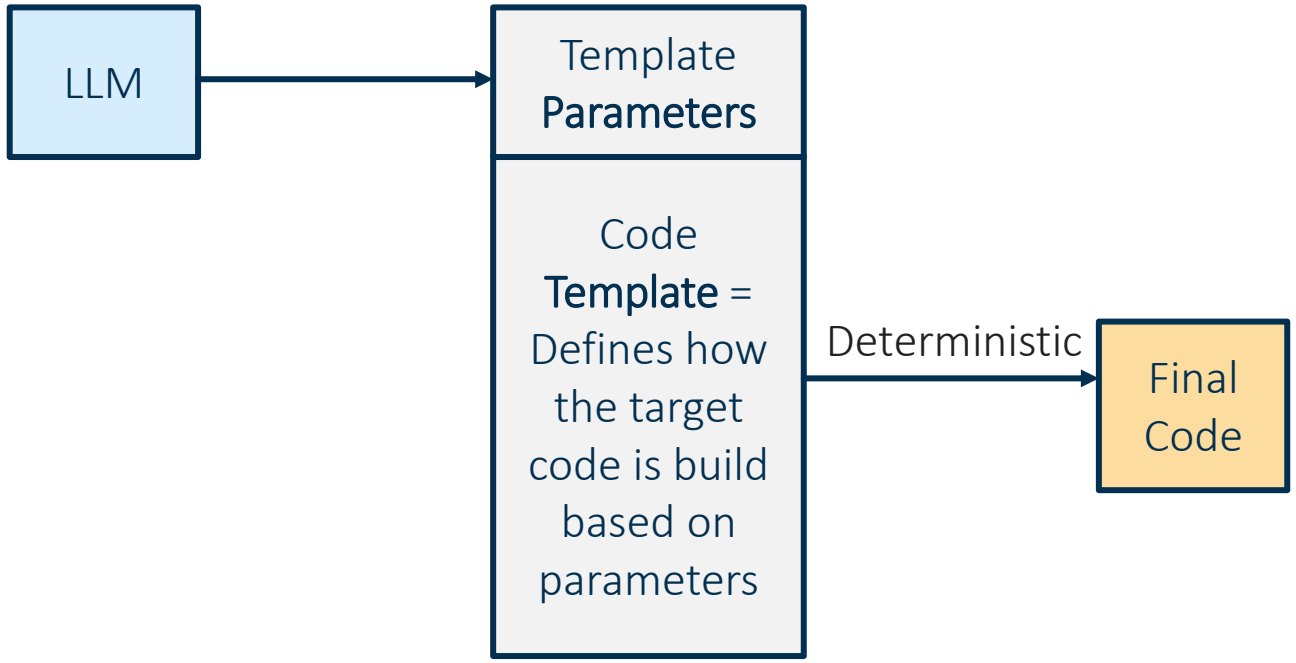
Different Ways to Generate the Code

Fully Deterministic Translation (A)

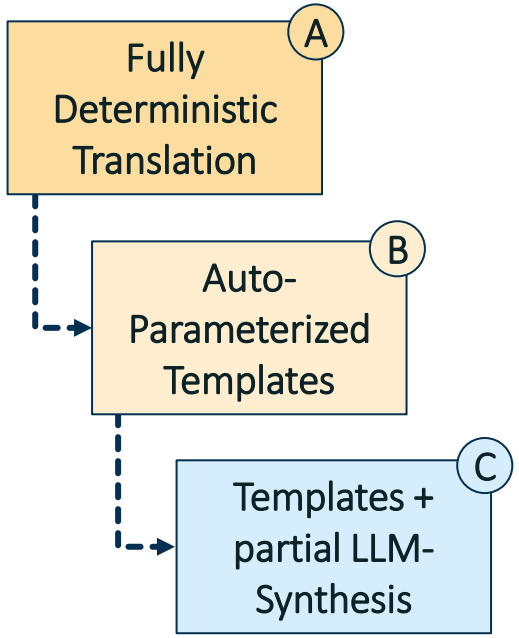


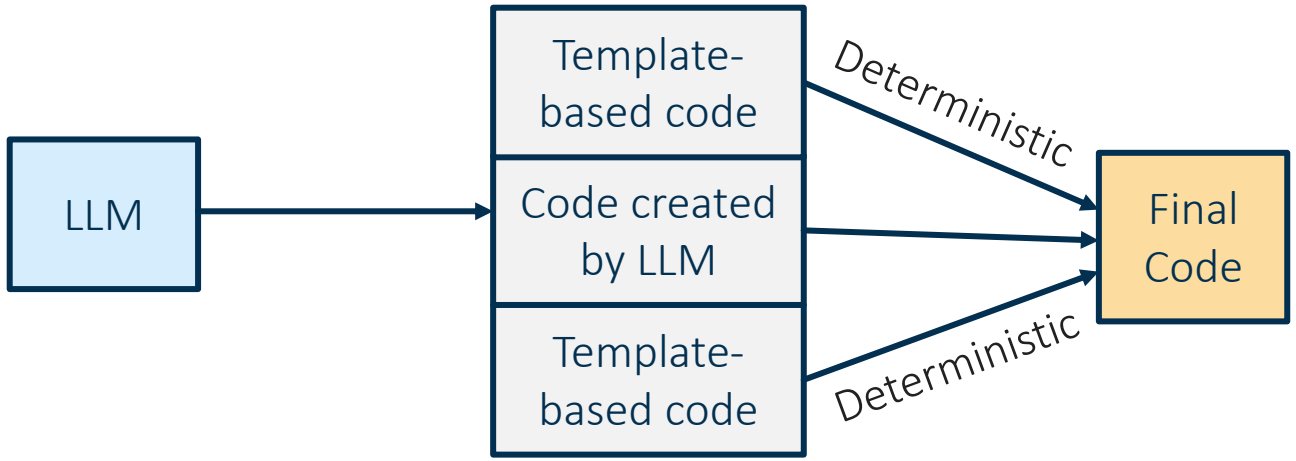
Different Ways to Generate the Code



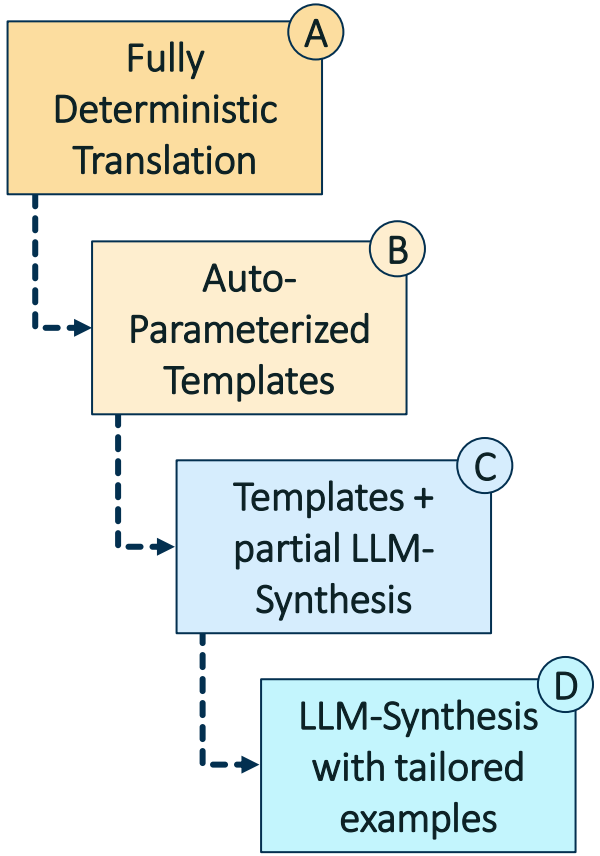


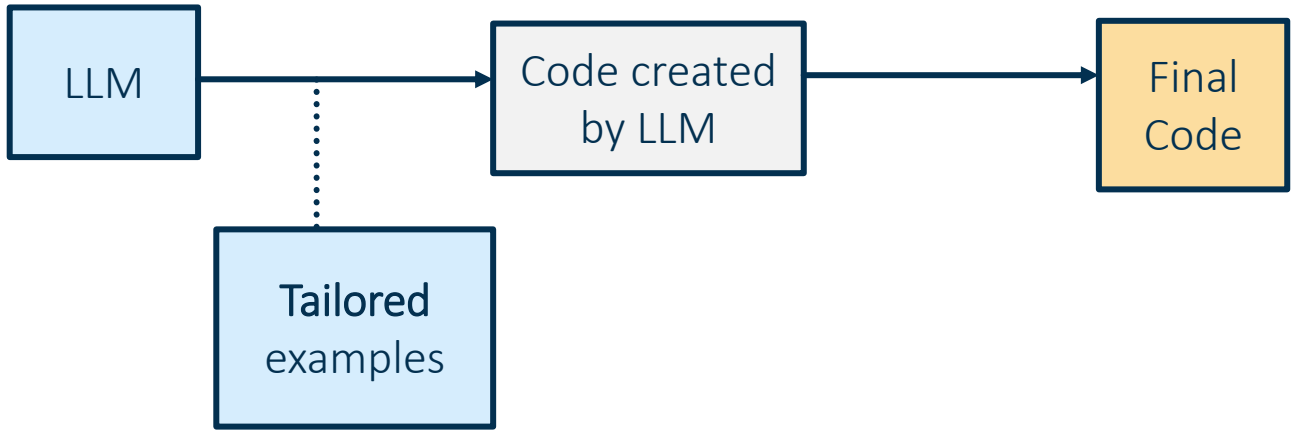
Different Ways to Generate the Code



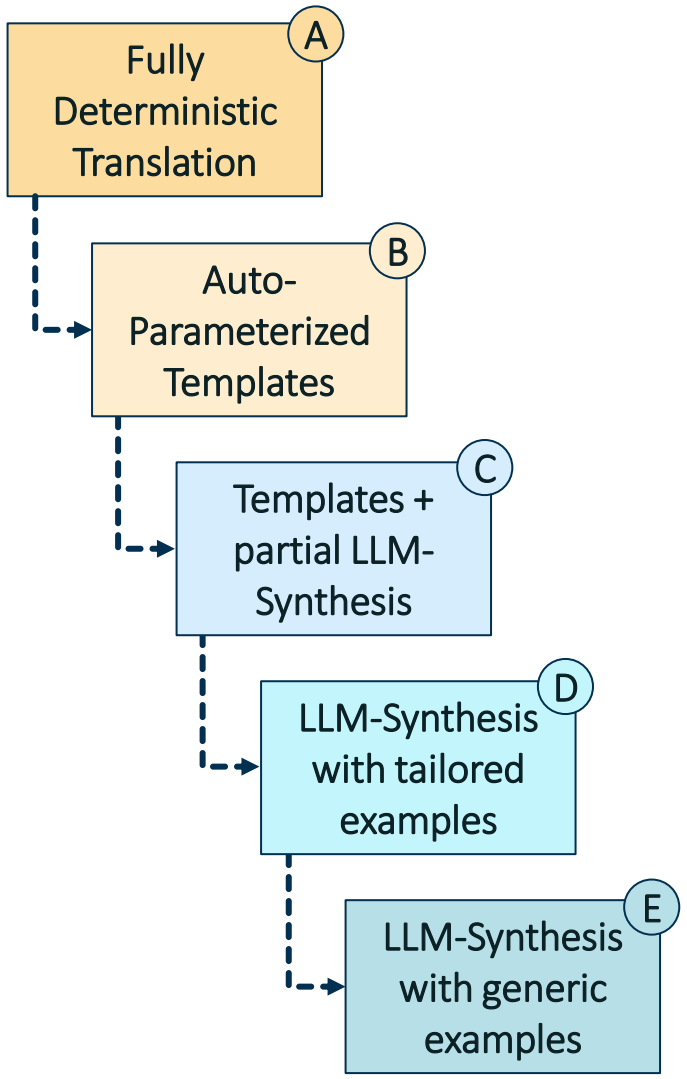


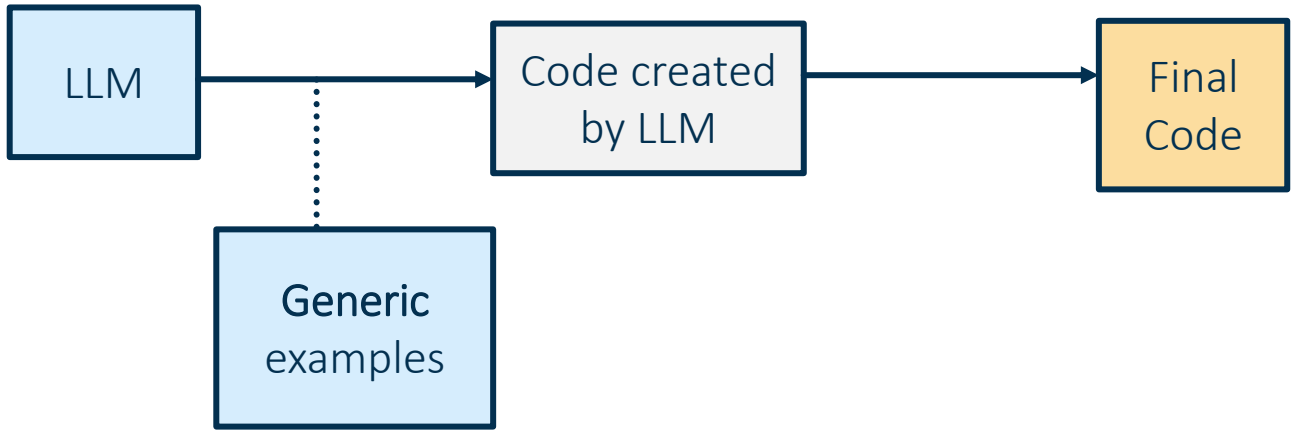
Different Ways to Generate the Code





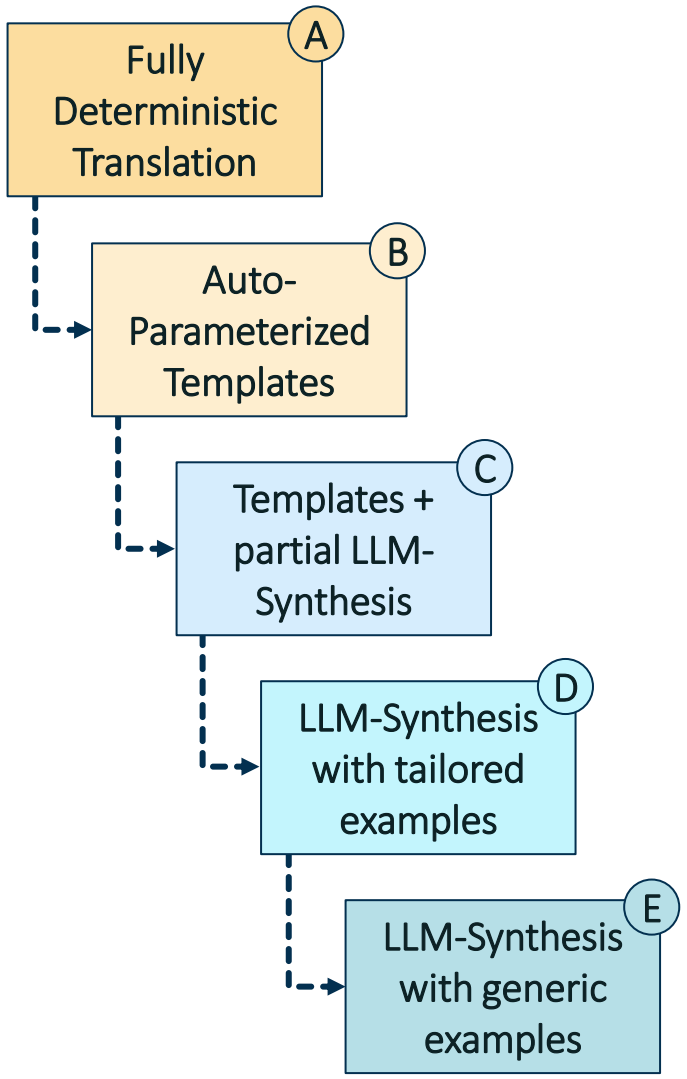
Different Ways to Generate the Code





Different Ways to Generate the Code

Use best fit based on code patterns & availability



A-Priori effort for writing templates, prompts, examples, …

Degree of freedom the LLM has and chance for **hallucinations**

Quality of the result in terms of code consistency and robustness

04 | Project Insights

Example I

Migration of an end-user business platform from SAS to Python/Jupyter

Area: **Insurance**
Status: **Finished**

KPIs

4k+ Files

1mio+ Lines of code

10 Macros

- Lots of ETL code; limited macro usage
- Hard requirements in terms of “getting the same results as before” → audit relevant code
- **>75% of the code** was migrated fully automatically into the final form, i.e., without the need for manual finetuning afterwards

Example II

Migration of a data processing pipeline from SAS to Python/PySpark

Area: **Publishing**
Status: **On-Going**

KPIs

450+ Files / Programs

100k+ Lines of code

200+ Macros

- Heavy macro usage
- Heavy usage of “enriched” SQL and custom formats
- Some descriptive statistical calculations
- We plan to achieve a degree of **automation of >70%** for this project

Example III

Migration of an SDTM processing pipeline for clinical trial data to R

Area: **Pharma**

Status: **Started recently**

KPIs

200+ Files / Programs

***k** Lines of code

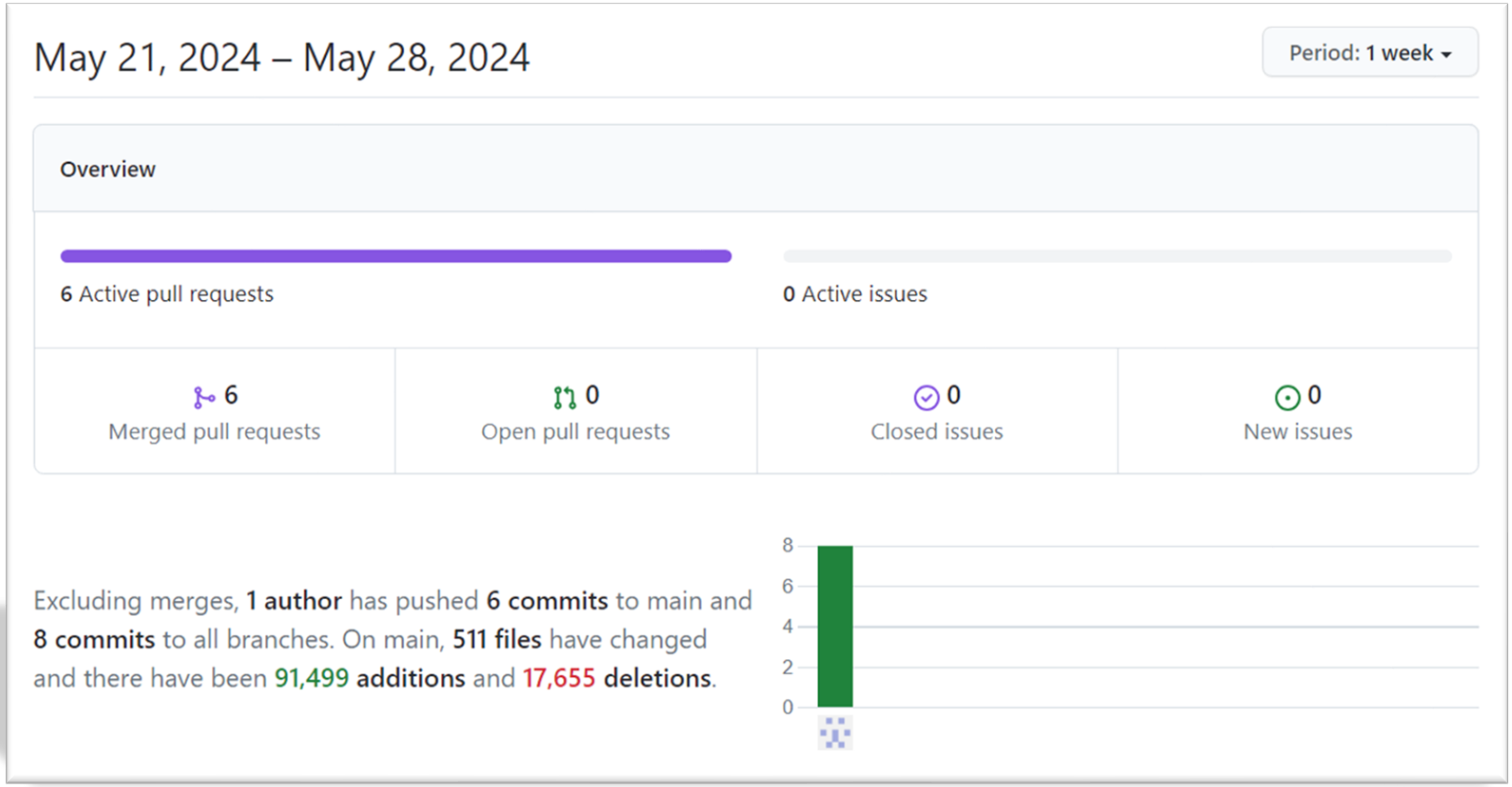
(currently under assessment)

150+ Macros

- Heavy macro usage
- SDTM pipelines; Some simple statistics
- Given that the first steps of our workflow are target architecture agnostic, we **heavily benefit from pre-existing processes / can directly start with R template development**
- Expected degree of automation: under assessment; But first results are promising!

HMS analytical software

The Power of Automation



Taken from a real project where we used our approach

Some Lessons Learned

Code Templates heavily support reliability

- No hallucinations
- No security risks
- No copyright issues

Checking correctness of translated code is a big cost factor

- Can easily be >50% of the total effort
- Benefits greatly from automatic testing (if possible)

Automatic Re-Execution is a game changer

- Effectively deal with requirement changes in on-going projects

Thank You!

Recap: What we discuss in this talk

- We developed an Expert Tool that **automates code translation from SAS to R/Python** in a reliable and robust way based on Generative AI and sophisticated templating
- Our approach gives us a **high degree of control over the translation output** which is crucial for clinical projects
- The approach significantly **accelerates our T&M projects** through faster completion and reduced costs: We used it in several SAS -> Python projects (insurance and finance sector) with great success and **substantial savings in terms of time and material**
- We recently started our **first SAS -> R migration project** for a large pharmaceutical company

Contact: robert.bauer@analytical-software.de

HMS analytical software

Contact me via
robert.bauer@analytical-software.de
if you are interested in this or similar topics!

Dr. Robert Bauer
Senior Data Scientist
HMS Analytical Software GmbH

HMS
analytical software