

## Code reference (SPARK)

This is a repositiorium with alpha version of the code prepared for peer-review.

The modules of SPARK agentic workflow are started from the 'main' files (see below in the #Agentic module starters).

All other scripts represent configs and helpers.

All development steps were performed using conda environment and Python 3.11.10.  
The full composition of environment including versions of all packages is provided in 'environment.txt' file.

## # Folder with agent and task DESCRIPTIONS.

### ***config/***

*\*agents.yaml*

Descriptions/definitions of agents.

*\*tasks.yaml*

Descriptions/definitions of tasks.

## # Task and pipeline CONFIGS

*\*config\_IG\_B.json*

Idea generation task: Use Case 1

*\*config\_IG\_C.json*

Idea generation task: Use Case 2

*\*config\_IG\_D.json*

Idea generation task: Use Case 3

*\*config\_COD.json*

Coding task

*\*config\_HUMP.json*

Human-triggered idea generation task

*\*config\_IREF.json*

Idea refinement task

*\*config\_PARAM\_VERIF.json*

Parameter verification task

*\*config\_PROGN\_EVAL.json*

Creating of database with parameter values for prognostic evaluation

## # Agentic module STARTERS

### ***\*\*Idea Generation Module***

*\*main\_IG\_B\_UC1.py*

Use case 1

*\*main\_IG\_C\_UC2.py*

Use case 2

*\*main\_IG\_D\_UC3.py*

Use case 3

### ***\*\*Idea Refinement Module***

*\*main\_IREF.py*

### ***\*\*Parameter Coding Module***

*\*main\_COD.py*

Comment: coding agentic module uses WSI analysis pipeline (for testing of and code improvement if necessary) from the folder 'cell\_pipeline' (see information below).

***\*\*Parameter Verification Module***

*\*main\_PARAM\_VERIF.py*

***\*\*Human-triggered idea processing module***

*\*main\_HUMAN\_PIPELINES.py*

**# HELPER scripts defining FLOWS within agentic modules**

*\*flow\_1b\_idea\_pipeline.py*

Flow for Idea Generation: Use Case 1 and 3

*\*flow\_1c\_idea\_pipeline.py*

Flow for Idea Generation: Use Case 2

*\*flow\_2\_coding.py*

Flow for Idea coding

**# HELPERS 1: agentic workflows**

*\*agent\_task\_init.py*

Helper script for initialization agents and tasks.

*\*crew\_init.py*

Helper script for crew initialization

*\*helper\_fn.py*

Helper script with multiple functions for the whole SPARK pipeline

*\*logging\_fn.py*

Helper script for logging

**# Pipelines for COHORT DATABASE PREPARATION (clinical data + parameter values) and correlation/prognostic analysis**

*\*main\_PROGN\_EVAL\_P1\_PREPARE\_DATABASE.py*

Part 1: Prepare one database per cohort containing clinical information and all parameter values (as derivatives)

*\*main\_PROGN\_EVAL\_P2\_ANALYSIS.py*

Part 2: Correlation analysis, Prognostic analysis (univariate and multivariate Cox analysis, Kaplan-Meier curves)

**# HELPERS 2: data preparation and analysis**

*\*X\_determine\_and\_process\_binary\_thresholds\_PART1.py*

Generate a table with absolute values of parameters for a row of quantiles.

*\*X\_determine\_and\_process\_binary\_thresholds\_PART2.py*

Generation of binarized thresholded area measurements

*\*X\_make\_merged\_csv.py*

Make csv file with values of all parameters at case level

*\*X\_transfer\_json\_to\_clear\_python\_code.py*

Transfer json-formatted Python snippet to clear Python code.

**# Folder with WSI ANALYSIS PIPELINE for PARAMETER TESTING within the CODING AGENTIC WORKFLOW**

***cell\_pipeline/***

*\*Cell\_Feature\_Functions.py*

Computes morphological and texture features (shape, curvature, grayscale properties) for individual cells from whole slide images.

*\*Evaluation\_Orders.py*

Defines and manages evaluation tasks and statistical summaries (mean, std, percentiles) for single-cell features across different cell types and tissue regions.

*\*MAIN\_Sparks\_Code\_Eval.py*

Main execution script that loads pre-computed WSI analysis objects and runs user-provided code snippets with time constraints and error handling.

*\*Mask\_Operations.py*

Handles tissue mask processing including scaling, buffering, border region detection, and conversion between different mask formats.

*\*Output\_Objects.py*

Manages output generation including CSV statistics, heatmap images, patch visualizations, and data serialization for WSI analysis results.

*\*Scaling\_Helper\_Functions.py*

Provides utility functions for coordinate transformations, image processing, statistical summaries, and visualization tasks across different resolution scales.

*\*Single\_Cell\_Objects.py*

Defines the single cell data structure that stores cell properties, polygon geometry, features, and neighborhood relationships.

*\*snippet\_fn.py*

Contains utility functions for executing user code with timeout constraints and logging results for the coding evaluation system.

*\*WSI\_Analysis\_Object.py*

Main analysis framework that coordinates whole slide image processing, mask overlay, patch generation, and single-cell feature extraction workflows.

### ***Comments and instruction for coding agentic workflow:***

1) SPARK idea coding was done with "anthropic/claude-3-5-sonnet-20241022".

2) You need to install the conda environment `CONDA_ENV_pathf_b_c_cleaned.yaml` (it is quite large to give the LLMs some freedom for their code). Within this environment, run the workflow "MAIN\_IDEA\_CODING.py 0" after some setup steps (see below).

3) This workflow selects a database from `"/coding_input_output/IDEA_DATABASE_CODING_INPUT"`, and starts coding all the ideas in the database one after another (other databases that were used for coding in `IDEA_DATABASE_B_C_REFINED`). Here, a small prepared test-database `"idea_database_small_selection_of_ideas.json"` is given for testing. The coding-agents and the coding-tasks are described in detail in `"config/agents"` and `"config/tasks"`.

4) The ideas in the database are coded from an LLM of your choice (selected in `CONFIG_CODING.json`) using the framework CREWAI, which supports the most common LLMs. The pipeline is tested and configured with LLMs from OPENAI and ANTHROPIC. You need a valid access key with some free money for coding, which you can pass in the `MAIN_Idea_Coding.py` under `OPENAI_API_KEY / ANTHROPIC_API_KEY`. Note that you can distribute the coding and code reviewing process to two different LLMs, but we recommend to have identical entries for `LLM_COD` and `LLM_COD_REV`.

5) After idea-coding, which produces a code-snippet and some metadata, this snippet is automatically executed within `"/cell_pipeline/MAIN_Sparks_Code_Eval.py"`. Since this happens in a separate python session, you need to define how your conda environment is loaded, check `"conda_env"` and `"CONDA_SOURCE_CALL"` in the `CONFIG_CODING.json` for correct setup on your machine (source call default works on most linux systems).

6) Before starting the pipeline, open the `./cell_pipeline/MAIN_Pathfinder.py` and define the path to the folder `"Coding_Prepared_Case_For_Fast>Loading/"`. This folder contains a compressed test case on which the code snippets are tested. After loading the test-setup (image, mask and cell-information), the new snippet is executed (with a timelimit that throws an error, default 60 seconds). If an error is raised, the snippet is passed to the review-agent (same as coding-agent but with additional error-message for debugging), after three negative attempts the idea is ultimately rejected. If the code runs successfully, it should create a tabular csv output and a `idea_s.json` (s for success, f for failed) in the folder `./coding_input_output/CODE_OUTPUT"`.