Supplement of Earth Syst. Dynam., 16, 423–449, 2025
https://doi.org/10.5194/esd-16-423-2025-supplement

Supplement of

Exploring the opportunities and challenges of using large language models to represent institutional agency in land system modelling

Yongchao Zeng et al.

Correspondence to: Yongchao Zeng (yongchao.zeng@kit.edu)

Supplementary Information I

CRAFTY Emulator

S1. Overview

The CRAFTY emulator, inspired by the methodology presented in Murray-Rust et al. (2014), shares the foundational methodology of the main CRAFTY model, emphasizing the microscopic interactions between Agent Functional Types (AFTs) that drive the emergence of key patterns, such as AFT changes and ecosystem service supply. The emulator employs essential data—such as AFTs, capitals, and demand—integral to the functionality of the model, without requiring modifications to the data structure.

Unlike the main CRAFTY model, the emulator is not intended to offer its full range of functionalities or replicate every detail of its outputs. Instead, the emulator serves as a streamlined environment for rapid testing and experimentation with new components that could potentially enhance the CRAFTY land system framework. Specifically, the emulator's AFTs base their decisions solely on economic benefits. This treatment intentionally excludes non-economic considerations, such as affinity for possessed land or social networks, reducing complicatedness and computational cost.

Despite its simplified nature, the emulator successfully reproduces key emergent patterns observed in the main CRAFTY model. At the microscopic level, land users respond to the utility provided by different ecosystem services. At the macroscopic level, the emulator demonstrates a tendency to drive the aggregated ecosystem supply to align with ecosystem demand, reflecting the system's adaptive dynamics. The simplifications in the emulator are designed not only to improve simulation speed and simplify parameterization but also to provide a more interpretable baseline. This baseline is particularly useful for observing the behaviour and influence of newly introduced components in a controlled and understandable setting.

Components tested in the emulator may or may not be integrated into the main model, depending on their added value and the evolving needs of ongoing projects.

This document outlines how the emulator is designed to facilitate the exploration of new components and provides an example to demonstrate its output similarities with the main model. Since the emulator does not introduce any new mechanisms compared to Murray-Rust et al. (2014), we omit discussion of its conceptual foundation and equations for brevity.

S2. Design

S2.1 Key Design Elements

S2.1.1 ModelState Interface

A central element of the emulator is the `ModelState` interface, which defines two methods: `setup` and `toSchedule()`. Any class that needs to be initialized and scheduled during a simulation must implement this interface.

S2.1.2 AbstractUpdater Class

Another critical component is the `AbstractUpdater` class, which depends on an `AbstractModelRunner` instance. The `AbstractUpdater` class implements both the `ModelState` and `Steppable` interfaces. The `Steppable` interface, provided by the MASON[1] library, designates objects that can execute step-by-step operations within the model. Classes with one-time operations can implement the `ModelState` interface directly. Conversely, classes requiring updates throughout the simulation need to extend the `AbstractUpdater` class.

S2.1.3 Simulation Flow Control with ModelRunner

The `ModelRunner` class is responsible for controlling the simulation flow. As a subclass of `AbstractModelRunner`, which extends the `SimState` class from the MASON library, `ModelRunner` is designed to have robust simulation management capabilities.

A notable feature of the `AbstractModelRunner` is the `stateManager`, an `ArrayList` that stores `ModelState` objects. Users can add `ModelState` instances to `stateManager` through the `loadStateManager` method, and the model runner will invoke their `setup` methods to initialize them in sequence. During execution, the MASON simulation engine detects and sequentially invokes the `step` method for each instance of `AbstractUpdater`.

This architecture has two major advantages:

a) Flexibility: Objects can be added to or removed from the `stateManager` easily, allowing for dynamic customization of the simulation.
b) Inter-Object Communication: The `stateManager` facilitates communication between different objects, enhancing modularity and integration across components.

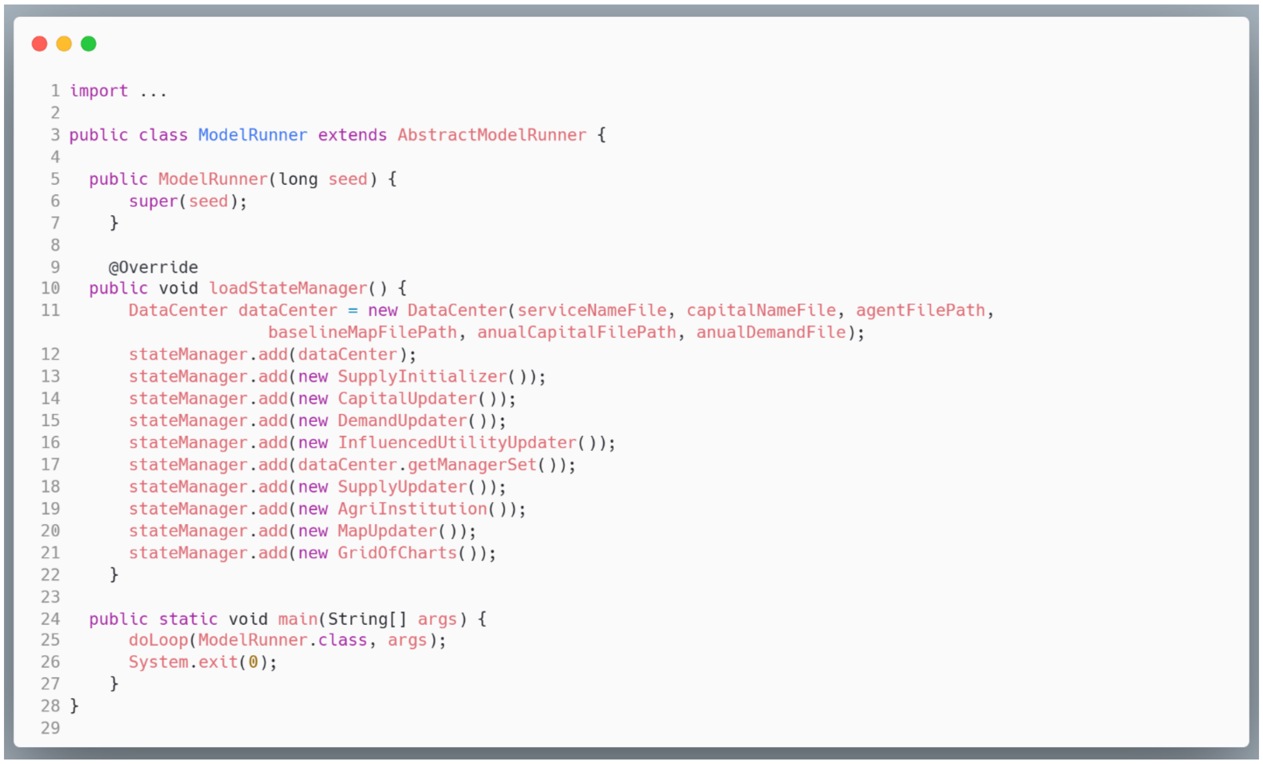
Figure S1 provides an example of the `ModelRunner` class, highlighting its structure and functionality. The class includes a constructor method and a `loadStateManager` method. In the `loadStateManager` method (lines 12 to 21), several instances are added to the `stateManager`. The names of these instances are related to their place in the workflow of the `ModelRunner`, and they are executed sequentially.

- The `DataCenter` instance, named `dataCenter`, is responsible for initializing and managing land-use data within the model.
- `SupplyInitializer` calculates the initial ecosystem service supply in CRAFTY.
- `DemandUpdater` updates the annual demand for ecosystem services.
- `InfluencedUtilityUpdater` computes the marginal utility for each ecosystem service.

[1] https://cs.gmu.edu/~eclab/projects/mason/

- `dataCenter`'s `getManagerSet` method returns a set of land managers who make land-use decisions.
- `SupplyUpdater` aggregates the ecosystem service supply from all land managers.
- `AgriInstitution` represents a policymaking agent that attempts to steer agricultural ecosystem services toward specific target levels.
- The `MapUpdater` and `GridCharts` components are used to visualize the AFT (Agent Functional Type) distribution and ecosystem service supply, respectively.

Certain lines, such as those involving the `AgriInstitution`, `MapUpdater`, and `GridCharts`, can be commented out without disrupting the simulation's functionality. This flexibility allows for easy integration/replacement of additional institutional agents (e.g., LLM institutional agents), similar to the approach used for `AgriInstitution`.



```java
import ...

public class ModelRunner extends AbstractModelRunner {

  public ModelRunner(long seed) {
      super(seed);
    }

    @Override
  public void loadStateManager() {
      DataCenter dataCenter = new DataCenter(serviceNameFile, capitalNameFile, agentFilePath,
                    baselineMapFilePath, anualCapitalFilePath, anualDemandFile);
      stateManager.add(dataCenter);
      stateManager.add(new SupplyInitializer());
      stateManager.add(new CapitalUpdater());
      stateManager.add(new DemandUpdater());
      stateManager.add(new InfluencedUtilityUpdater());
      stateManager.add(dataCenter.getManagerSet());
      stateManager.add(new SupplyUpdater());
      stateManager.add(new AgriInstitution());
      stateManager.add(new MapUpdater());
      stateManager.add(new GridOfCharts());
    }

  public static void main(String[] args) {
      doLoop(ModelRunner.class, args);
      System.exit(0);
    }
}

```

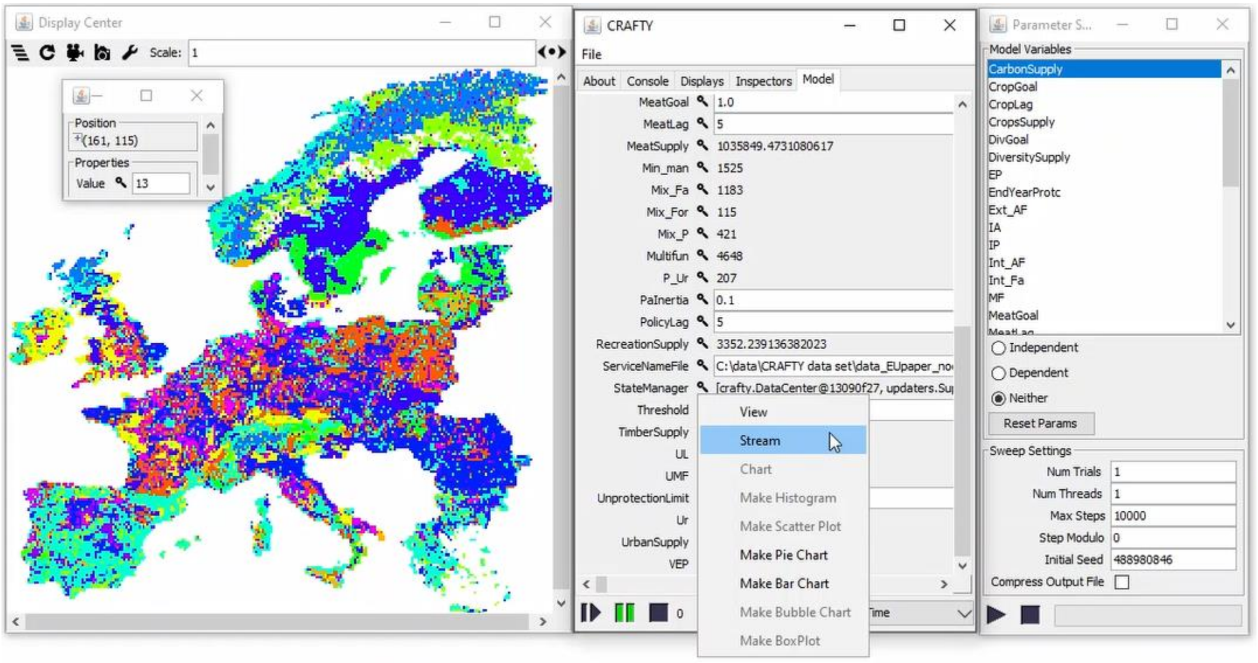
Figure S1. An example implementation of the ModelRunner class

S2.2 Graphic user interface (GUI)

Instances of the `ModelRunner` class can be executed directly. However, this approach lacks the flexibility needed for exploring and interacting with the model, both of which are crucial for effective model development and usage. To address this limitation, the MASON library, used in the emulator, provides a comprehensive GUI. This GUI allows users to visualize model data, explore parameter spaces, and set up experiments. For an example of how a `ModelRunner` instance (e.g., instantiated as `new Intra()` in the source code) is configured to enable the GUI, refer to the `ModelRunnerWithUI` class in the `display` package.

Figure S2 illustrates several components of the GUI:

- Map Window: Displays a real-time updated map. Users can click on the map to view detailed information about individual land cells.
- Central Window: Contains multiple tabs, with the "Model" tab displaying the model's parameters and objects. By clicking on the key symbols, users can visualize parameters, stream data, or drill down into objects to inspect their fields. For instance, clicking the key symbol next to `stateManager` and selecting "View" will show a list of objects that, by design, contain nearly all the model's information.
- Parameter Sweeping Window: The rightmost window provides an interface for setting up parameter sweeping experiments.



Position
(161, 115)
Properties
Value 13

MeatGoal	1.0
MeatLag	5
MeatSupply	1035849.4731080617
Min_man	1525
Mix_Fa	1183
Mix_For	115
Mix_P	421
Multifun	4648
P_Ur	207
PaInertia	0.1
PolicyLag	5
RecreationSupply	3352.239136382023
ServiceNameFile	C:\data\CRAFTY data set\data_EUpaper_no
StateManager	[crafty.DataCenter@13090f27, updaters.Su
Threshold	
TimberSupply	
UL	
UMF	
UnprotectionLimit	
Ur	
UrbanSupply	
VEP	

Model Variables
CarbonSupply
CropGoal
CropLag
CropsSupply
DivGoal
DiversitySupply
EP
EndYearProtc
Ext_AF
IA
IP
Int_AF
Int_Fa
MF
MeatGoal

Independent [✓ Neither] Dependent

Sweep Settings

Num Trials	1
Num Threads	1
Max Steps	10000
Step Modulo	0
Initial Seed	488980846
Compress Output File	☐

Figure S2. GUI of the emulator.

S3. Output comparison

As mentioned above, the emulator shares substantial conceptual similarities with the main CRAFTY model and is compatible with the input data used by the main model. These contribute to the resemblance in their outputs. To demonstrate their output similarity, we use the dataset of RCP2.6-SSP1 from https://osf.io/vfjsn as shared input for both the emulator and the main model. This dataset is also used to configure the experiments in this paper, which includes files indicating the temporal changes in capital distribution, a file listing AFT names and indices, a set of files specifying the parameters of each AFT, and a file detailing the changes in demand for different ecosystem services.

A key feature of the outputs is the tendency to align ecosystem service supply with demand. As illustrated in Fig. S3 (data available at Zeng (2025a)), with the exception of timber supply, the supply of all other services (represented by black scatter points for the emulator and blue solid curves for the main model) shows a clear tendency to converge toward the demand (depicted as red solid curves). The deviation of timber supply from demand is due to a decline in the capitals associated with timber production over time, which reduces the resources available to AFTs, thereby limiting their ability to meet the demand for timber.

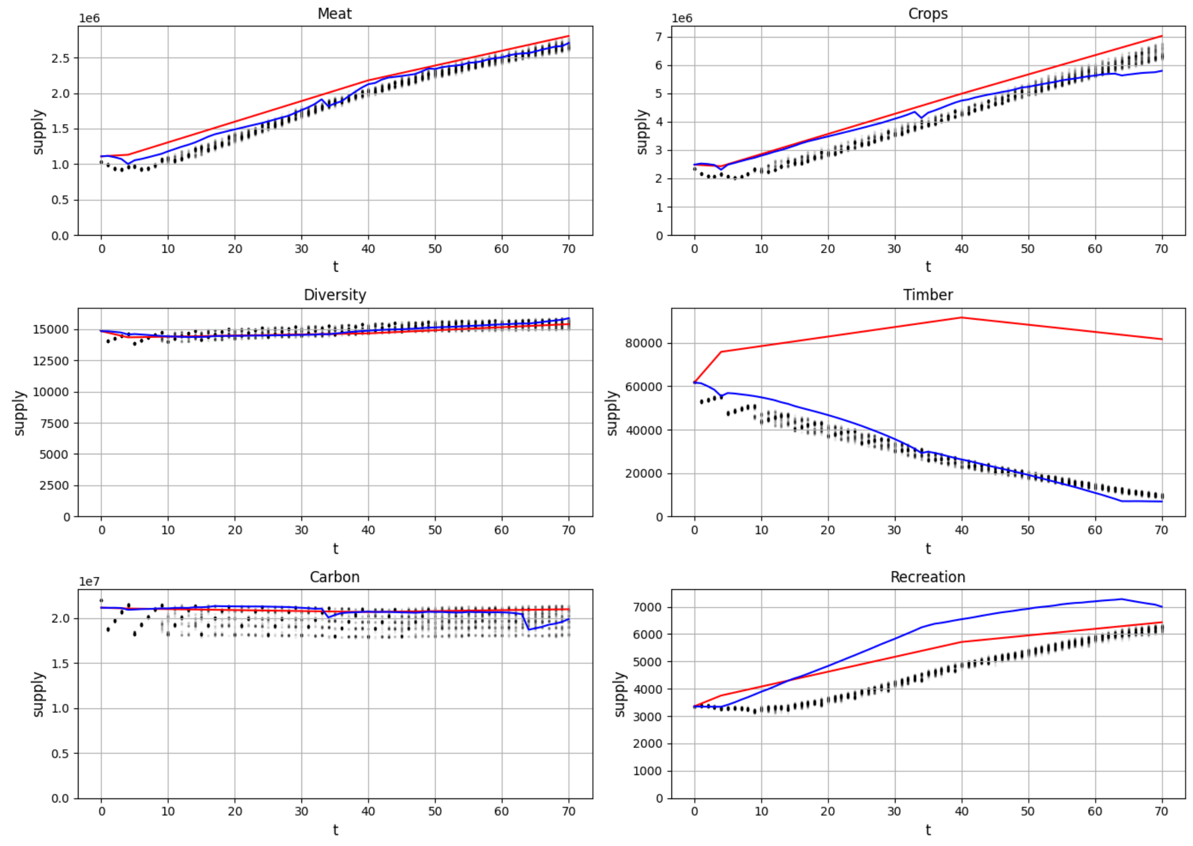


Figure S3. The output ecosystem service supply of the emulator and CRAFTY main model. Red curves are demand; blue curves are output from the main model; the dots are results from 100 times repeated simulations using the emulator.

In Fig. S4, the numbers of AFTs at the end of simulations (the 70th iteration) also show a high output resemblance. Figure S5 shows the number of AFTs relative to the baseline data shared by both the emulator and the main model. It can be seen that the relative AFT numbers from the emulator and main model are quantitatively different. For example, the results from the emulator include more Ext_AF and Int_Fa but less Int_AF, IP, UL and MF than the main model outputs. However, the changing directions of the AFTs are identical.

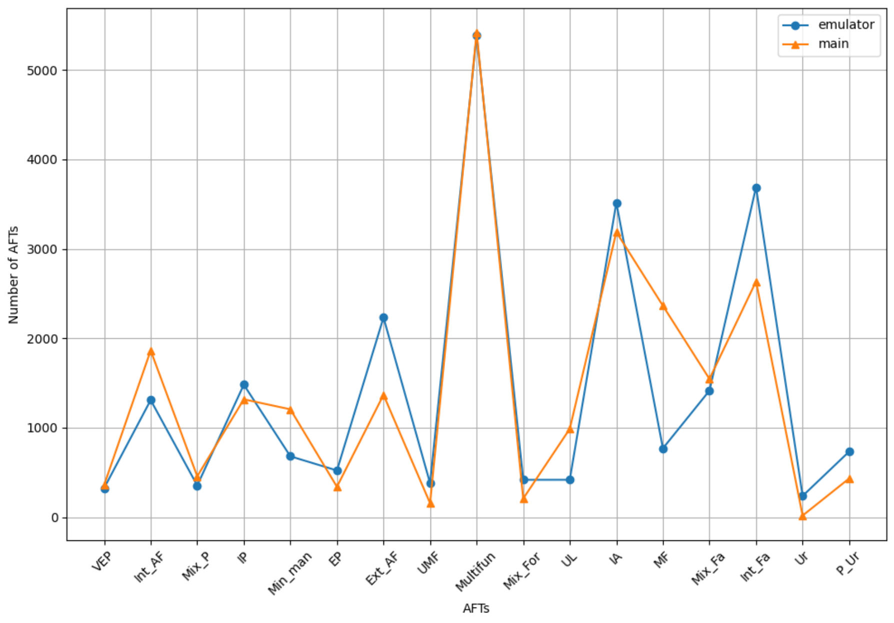


Figure S4. Resultant AFT numbers of the emulator and main model. The results from the emulator are averaged across 100 repeated simulations.

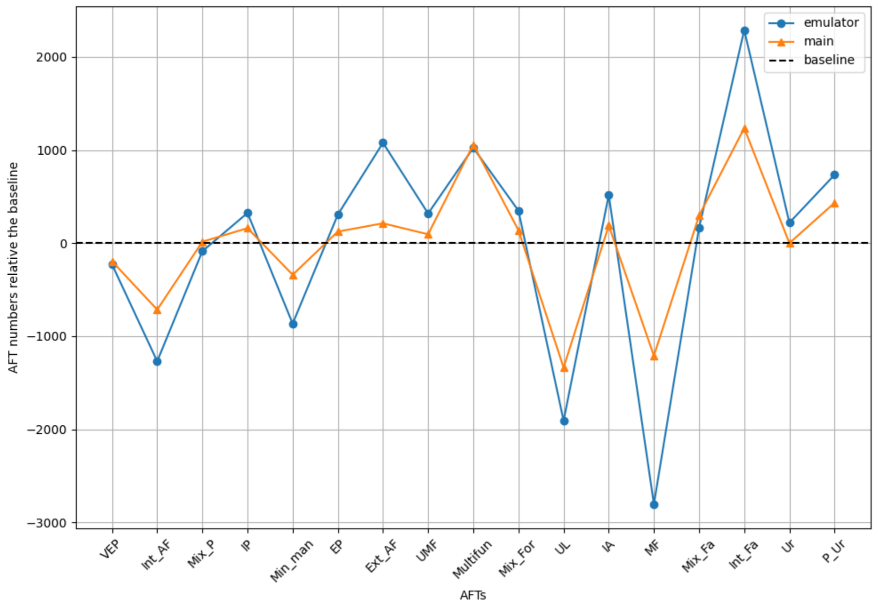


Figure S5. AFTs numbers relative to the baseline data shared by the emulator and main model. The results from the emulator are averaged across 100 repeated simulations.

A key distinction between the emulator and the main model is that the emulator does not consider social factors. To make a fair comparison of outputs, we disabled the social processes in the main model. However, other differences remain that can contribute to variations in the results. For instance, the emulator does not incorporate land abandonment—a mechanism that allows AFTs to temporarily occupy fewer land cells—or the give-in/up threshold, which increases the difficulty of land turnover. These omissions can lead to more pronounced land-use changes in the emulator compared to the main model.

This disparity is noticeable in Fig. S5, where the numbers of AFTs fluctuate more notably in the emulator than in the main model. However, both land abandonment and the give-in/up threshold are deeply integrated into the main model's framework, and their removal would require code refactoring. The development team decided against refactoring the main model to maintain compatibility for comparison purposes, as the emulator's behaviour is considered acceptable for its exploratory use.

It is important to note that the emulator's tendency to align supply with demand is driven by the micro-mechanisms inherent in the CRAFTY framework. This clear and simplified dynamic is intentional, as it provides a well-defined benchmark for observing and understanding how newly added components (e.g., institutional agents) influence system behaviour. In contrast, the main model incorporates richer mechanisms that can produce more complex behaviours, which may often deviate from alignment between supply and demand. This added complexity enhances the main model's descriptive power and makes it better suited for capturing the complexity of real-world dynamics.

Supplementary Information II

Application of Genetic Algorithm for Optimal Policy Search

The optimal policy actions were determined using the NSGA-II genetic algorithm (Deb et al., 2002), implemented through the Rhodium Python library (Hadjimichael et al., 2020). To integrate the genetic algorithm with the CRAFTY emulator (developed in Java), the Py4J library (Py4J, 2025) was utilized for communication between the two environments (source code available at Zeng (2025b)). The diagram in Fig. S6 illustrates the process.

The search begins by instantiating the CRAFTY model on the Java side, creating a fully functional CRAFTY object ready to be invoked from Python. On the Python side, the program initializes by randomly generating a population of policy action time series. Each time series is represented as a list of 21 integer elements with values constrained to the range [-5, 5], corresponding to different tax change levels for policy actions. Each simulation was run for 110 iterations. Every five iterations (starting from the 0th iteration), the institutional agent updates its policy action by using the next integer in the list.

These generated policy action time series are passed as arguments to the CRAFTY object via a Python wrapper function. The CRAFTY model processes each series, iterating through the specified policy actions and producing a corresponding time series of meat supply outputs. These outputs are then evaluated using the least squares method to calculate the total error relative to the predefined policy goals. This total error serves as the objective that the NSGA-II algorithm seeks to minimize.

After computing the error, the program checks if the maximum number of algorithm iterations (set to 1000 in this experiment) has been reached. It is important to note that these iterations refer to the optimization process performed by the NSGA-II algorithm and are distinct from the internal iterations within the CRAFTY model itself. If the maximum number of algorithm iterations has not been reached, the NSGA-II algorithm generates a new set of policy action time series (offspring variants) for further evaluation. This iterative process continues until the stopping condition is met.

The optimal policy actions and the corresponding final total error are saved in a JSON file for analysis.

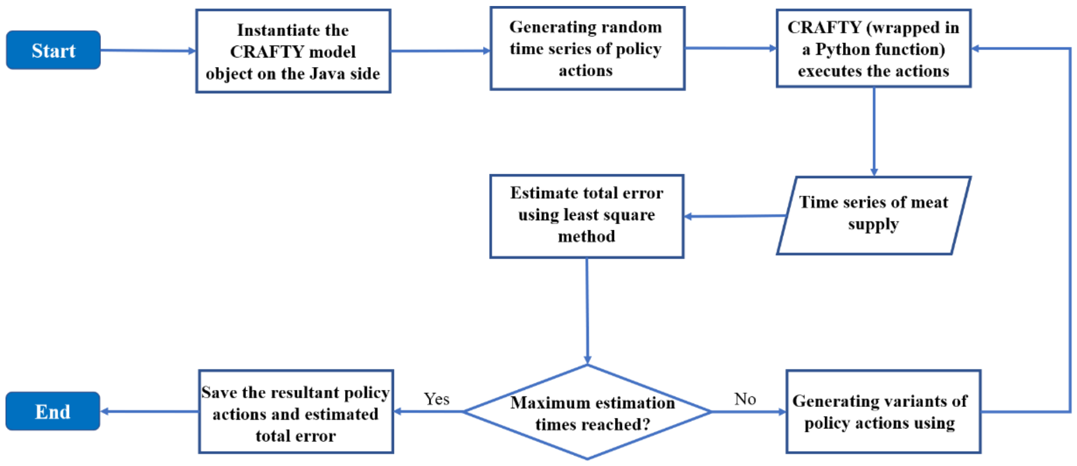

Start
Instantiate the CRAFTY model object on the Java side
Generating random time series of policy actions
CRAFTY (wrapped in a Python function) executes the actions
Time series of meat supply
Estimate total error using least square method
Maximum estimation times reached?
Yes
No
Save the resultant policy actions and estimated total error
Generating variants of policy actions using
End

Figure S6. The process of applying the NSGA-II algorithm to search optimal policy actions within the CRAFTY emulator.

Supplementary Information III

Prompt refinement

Table S1 juxtaposes the initial draft and the final version of the prompt. We started with the prompt of Agent S2 because it saves historical interactions with the human user and can use past experience to improve its output, which is necessary for human-in-loop prompt improvement. The prompts of Agent S1.1 and Agent 1.2 are derived as variations of the Agent S2 prompt. Table S2 highlights the key differences between the initial draft and the final prompts in Table S1. The changes mainly involve simplifying and clarifying the tasks for the agents instead of prescribing decision-making strategies or policy preferences. A time-intensive refinement is to explain the format of the output and to add notes to fix observed unwanted behaviours.

Table S1. The initial draft and final version of the prompt for Agent S2

Initial draft of prompt for Agent S2	**Final version of prompt for Agent S2**
Context: In a land use change simulation, you are playing the role of a policymaker who intends to influence meat production using economic policies including subsidies and taxes. During the simulation, you will be updated on information regarding historical meat supply and demand. You also have access to the policies you used in the past. You will be offered the average gap between the prescribed policy goal and actual meat production across the recent five years, which is labelled as "AveGap" below. You use the information in a way that you think is reasonable to conduct policy adaptation periodically to close the gap. Your output consists of two parts: one part is the reason explaining how you make decision; the other part is a number representing the intensity of policy intervention you are going to implement. Please note that your step-wise reasoning is normally more reliable and reasonable. You will also be provided with a summary of the reason how you made a policy decision last time as a reference to reflect your decision-making. Information: 1. Meat supply [values of meat supply in temporal order] 2. Meat demand [values of meat demand in temporal order] 3. AveGap: a number 4. Policy intervention [values of historical policy intervention] Summary of Policy decision reasoning last time: {summary here}	Simulation Role: Assistant to Economic Policymaker in Land Use Change Scenario. Objective: Develop tax policies to effectively manage meat production, aligning with set policy goals. Policy Tools: Taxes for regulating meat production levels. Information Provided: 1. General Context: As an assistant, propose tax-based policies for meat production management. Interaction with policymaker is crucial for refining decisions and gaining your experience in policymaking. 2. Data: - Policy goal: {policy_goal} - Average error (avg_err):{avg_err}. - Historical policy actions: {hist_actions} 3. Recent interaction with policymaker: {convers} 4. Experience: {exp} Guidance for Decision-Making: - Use historical data and policymaker feedback for policy adjustments. - Aim to minimize the absolute value of avg_err. - Provide logical, sequential reasoning. - Reflect on experience for current decision enhancement. Interaction Instructions:

	1. Review historical information, recent interaction with policymaker, and your experience. 2. Assess the impact of previous policies. 3. Develop your policy rationale in a step-by-step manner. 4. Propose a specific policy action. Required Output Format: 1. Proposal Reasoning: [Your explanation] 2. Policy Action Proposal Without Reasoning: - Indicate your proposed tax policy change using symbols and numbers. - Use '+' to signify an increase in tax levels, '-' for a decrease, and '0' to maintain the current level. - Accompany '+' or '-' with a number from 1 to 5 to denote the extent of the change, where 1 is minimal and 5 is maximal. - Examples: "+3" for a moderate increase, "-1" for a slight decrease. - If proposing to maintain the current tax level ('0'), no additional number is needed. - Surround the proposed action using a pair of hashtags [Indicate your proposal here, e.g., "#+3#", "#-2#", "#0#"] Here are three examples to show you the format to output Policy Action Proposal Without Reasoning: 1. Policy Action Proposal without reasoning: "#-1#" 2. Policy Action Proposal without reasoning: "#+3#" 3. Policy Action Proposal without reasoning: "#-5#" Note: Always specify a clear policy action. If uncertain, propose a tentative action based on available data. Don't fake interaction with policymaker if there is no interaction yet. avg_err > 0 means meat undersupply, while avg_err < 0 means meat oversupply.

Table S2. Key differences between the initial draft and final version of the prompt for agent S2

Key aspects	Draft	Final version	Reason for changes
Roles	Assigning the agent the role of a policymaker directly.	Shifting the role from a policymaker to an assistant	Using the draft prompt, the agent was not able to act as expected. Using human-in-loop prompt improvement needs to define the agent as an assistant, whose proposals need to be checked by the human operator. Using the final prompt, the human operator's response is set to approve every proposal from the

			agent by default to enable full autonomy.
Action scope	Using taxes and subsidies.	The focus narrows to tax-based policies for regulating meat production.	Simplify the scope of policy actions to improve output reliability.
Information	Meat demand and supply are included.	Meat demand and supply are omitted.	Reduce information redundancy to avoid the LLM agents attempting to align meat supply with demand.
Target	A general guidance on using historical data to adapt policies and close the gap (AveGap) between goals and actual production.	Clearly defines the objective as minimizing the absolute value of avg_err (error between goal and actual production)	Explaining the target more explicitly.
Structure	Structure is less formalized	Information is organized into several sections to clearly delineate objectives, tasks, and required output.	Making the prompt friendlier for improvement.
Output formatting	Lack of detailed formatting guidelines.	A specific, symbolic format and example to detail the structure of output.	This is very important and takes much time to adjust to achieve a reliable output structure that can be parsed by the emulator.
Notes	Without notes.	With notes such as “do not fake interaction with policymaker” and “proposing a tentative action” if uncertain.	The notes are intended to fix erroneous behaviours of the LLM agents observed during the prompt improvement process.

Table S3 shows the initial prompt draft and final prompt for Agent Q. The prompt refinement process for Agent Q was relatively simple because the part of the prompt that indicates the required output format could be taken from the final prompt for Agent S2. In addition to this, there are some key improvements worth noting. Instead of providing Agent Q with the average error, we found that the policy goal seemed more straightforward for Agent Q. Agent Q’s multi-role conversations also resulted in less misunderstanding of its target, even with meat demand and supply provided. This might be because the conversations between different roles break down the given problem into smaller components during role-specific information exchange, which is more manageable for LLMs (Wei et al., 2022). To streamline a smooth dialogue, an example is given. This example works as a guideline rather than a rule that confines the agent’s conversation generation. The added “Notes” at the end of the final prompt are aimed at fixing issues observed during the test. For instance, the agent occasionally used a pair of hashtags in the middle of a dialogue to highlight an undecided policy action, which should be avoided. Additionally, the agent sometimes forgot to include the sign (+/-) of a policy action, leading to ambiguity. These refinements were implemented to enhance accuracy and consistency in the agent’s outputs.

Table S3. Initial draft and final version of prompts for Agent Q

Initial draft of prompt for Agent Q	Final version of prompt for Agent Q
Your task: Generate a believable conversion among different roles relevant to tax policies for meat production.	Engage in a role-playing conversation about tax policies affecting meat production, integrating data analysis and diverse perspectives.

Information provided: 1. Data: - Historical Policy Actions (adapted every five years): {hist_actions} - Historical meat demand averaged every five years: {meat_demand} - Historical meat supply averaged every five years: {meat_supply} - Historical errors between meat supply and policy goal (avg_err): { avg_err } 2. Roles and responsibility - Policy Analyst: Review the data provided, and interpret the data to start the conversation. Policy Analyst should always note that avg_err greater than 0 means meat undersupply, while avg_err less than 0 means oversupply. - Government Official: Aiming at maintaining avg_err at 0. Government Official will listen to others' opinions, justify own judgement and make policy decision regarding tax adjustment for meat production. For simplicity, the official can only increase, decrease, or maintain the tax level by a reasonable level ranging from 0 to 5, six levels in total. - Economists: Conduct a cost-benefit analysis of the proposed policies. Assess the economic impact on the budget, taxpayers, and overall economy. Identify any potential economic risks or opportunities. - Meat producer representative: Represent the views and concerns of the meat producers affected by the policy. Provide feedback on how the policy will impact the meat producers, and suggest modifications to better serve their needs. - Environmentalist: Concerned with the environmental issues caused by meat production, and suggest policy adjustments benefiting environmental protection. Required output and format: Conversation: place the generate conversation here. Policy action: place a clear policy action here using a pair of hashtags to highlight the policy action.	**Background Data:** - **Historical Policy Actions** (updated every five years): {policy_actions} - **Meat Demand ** (averaged every five years): {meat_demand} - **Meat Supply** (averaged every five years): {meat_supply} - **Policy goal** maintain the meat production at: {policy_goal} **Roles & Responsibilities:** 1. **Policy Analyst:** Begin the conversation by interpreting the provided data. 2. **Government Official:** Strive to achieve policy goal. Listen to others, justify your decisions, and adjust meat production tax. 3. **Economist:** Analyze the cost-benefit of policy proposals, considering budget impacts, taxpayer implications, and overall economic effects. Highlight risks and opportunities. 4. **Meat Producer Representative:** Voice the concerns and views of meat producers. Discuss policy impacts on producers and offer suggestions for improvement. 5. **Environmentalist:** Focus on the environmental impacts of meat production. Propose policy adjustments for environmental protection. **Required Output & Format:** - **Conversation Flow:** Engage each role in a structured dialogue, reflecting their unique perspectives and data interpretation. - **Policy Action:** Extract the final policy action from the conversation and output it in required format below: - Indicate the official's policy action using symbols and numbers. - Use '+' to signify an increase in tax levels, '-' for a decrease, and '0' to maintain the current level. - Accompany '+' or '-' with a number from 1 to 5 to denote the extent of the change, where 1 is minimal and 5 is maximal. - Examples: "+3" for a moderate increase, "-1" for a slight decrease. - If proposing to maintain the current tax level ('0'), no additional sign is needed.

- Indicate the official’s policy action using symbols and numbers. - Use '+' to signify an increase in tax levels, '-' for a decrease, and '0' to maintain the current level. - Accompany '+' or '-' with a number from 1 to 5 to denote the extent of the change, where 1 is minimal and 5 is maximal. - Examples: "+3" for a moderate increase, "-1" for a slight decrease. - If proposing to maintain the current tax level ('0'), no additional sign is needed. - Surround the proposed action using a pair of hashtags Here are three examples to show the format to output Policy Action: 1. "#-1#" 2. "#+3#" 3. "#-5#"	- Surround the proposed action using a pair of hashtags Here are three examples to show the format to output Policy Action: 1. "#-1#" 2. "#+3#" 3. "#-5#" <u>**Example Dialogue Structure:**</u> <u>1. Policy Analyst provides data summary and initial observations.</u> <u>2. Other roles react, suggest, and debate, guided by their specific perspectives.</u> <u>3. Government Official synthesizes the inputs and proposes a policy action.</u> <u>4. Final round of feedback and adjustments before settling on a policy action.</u> <u>Note:</u> <u>Do not use hashtags in the dialogue. Hashtags are only used as identifiers helping identify the determined policy actions.</u> <u>Important: "+" means increase tax; "-" means decrease tax.</u>

Supplementary Information IV

Settings for the CRAFTY emulator

In this study, CRAFTY was initialized by allocating AFTs, capital maps, and demand parameters in accordance with a designated Representative Concentration Pathway (RCP) (Van Vuuren et al., 2011) and Shared Socioeconomic Pathway (SSP) (O'neill et al., 2014). We employed the version of CRAFTY that was previously calibrated using outputs from the IMPRESSIONS Integrated Assessment Platform (IAP) (Harrison et al., 2015; Holman et al., 2017). The IAP is a cross-sectoral, multi-model framework widely used to explore European land system changes and has been rigorously evaluated in various studies (Brown et al., 2015; Harrison et al., 2016; Kebede et al., 2015; Pedde et al., 2019). By relying exclusively on IAP-derived input data, we ensure that both socio-economic and climatic drivers are implemented in a consistent manner and that any observed land-use transitions are attributable to either CRAFTY's internal dynamics or the conditions imposed by each scenario. More specific details on calibration procedures for this model version can be found in Brown et al. (2019).

For the simulations presented here, the model was run at a 10 arcminute (10') resolution, which yields 23,871 grid cells. This resolution matches the granularity of the available input data, offers manageable computational demands, and is aligned with the current state of calibration data availability. Although CRAFTY can be applied to various scenarios derived from different RCP and SSP combinations, in this paper we focus on SSP1–RCP2.6. This particular scenario assumes a relatively modest level of climate change, coupled with gradually improving socio-economic conditions such as steady economic growth, effective governance, strong social cohesion, and robust international collaboration. We acknowledge that exploring multiple RCP–SSP combinations would provide additional insights into the variability and uncertainty across scenarios, and future research will aim to extend this work to a broader range of climate and socio-economic pathways.

Reference

Brown, C., Seo, B., and Rounsevell, M.: Societal breakdown as an emergent property of large-scale behavioural models of land use change, Earth Syst. Dynam., 10, 809–845, https://doi.org/10.5194/esd-10-809-2019, 2019.
Brown, C., Brown, E., Murray-Rust, D., Cojocaru, G., Savin, C., and Rounsevell, M.: Analysing uncertainties in climate change impact assessment across sectors and scenarios, Climatic Change, 128, 293-306, https://doi.org/10.1007/s10584-014-1133-0, 2015.
Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on evolutionary computation, 6, 182-197, https://doi.org/10.1109/4235.996017, 2002.

Hadjimichael, A., Gold, D., Hadka, D., and Reed, P.: Rhodium: Python library for many-objective robust decision making and exploratory modeling, Journal of Open Research Software, 8, https://par.nsf.gov/servlets/purl/10314245, 2020.
Harrison, P. A., Holman, I. P., and Berry, P. M.: Assessing cross-sectoral climate change impacts, vulnerability and adaptation: an introduction to the CLIMSAVE project, Climatic Change, 128, 153-167, https://doi.org/10.1007/s10584-015-1324-3, 2015.
Harrison, P. A., Dunford, R. W., Holman, I. P., and Rounsevell, M. D. A.: Climate change impact modelling needs to include cross-sectoral interactions, Nature Climate Change, 6, 885-890, https://doi.org/10.1038/nclimate3039, 2016.
Holman, I. P., Brown, C., Janes, V., and Sandars, D.: Can we be certain about future land use change in Europe? A multi-scenario, integrated-assessment analysis, Agricultural Systems, 151, 126-135, https://doi.org/10.1016/j.agsy.2016.12.001, 2017.
Kebede, A. S., Dunford, R., Mokrech, M., Audsley, E., Harrison, P. A., Holman, I. P., Nicholls, R. J., Rickebusch, S., Rounsevell, M. D. A., Sabaté, S., Sallaba, F., Sanchez, A., Savin, C., Trnka, M., and Wimmer, F.: Direct and indirect impacts of climate and socio-economic change in Europe: a sensitivity analysis for key land- and water-based sectors, Climatic Change, 128, 261-277, https://doi.org/10.1007/s10584-014-1313-y, 2015.
Murray-Rust, D., Brown, C., van Vliet, J., Alam, S. J., Robinson, D. T., Verburg, P. H., and Rounsevell, M.: Combining agent functional types, capitals and services to model land use dynamics, Environmental Modelling & Software, 59, 187-201, https://doi.org/10.1016/j.envsoft.2014.05.019, 2014.
O'Neill, B. C., Kriegler, E., Riahi, K., Ebi, K. L., Hallegatte, S., Carter, T. R., Mathur, R., and van Vuuren, D. P.: A new scenario framework for climate change research: the concept of shared socioeconomic pathways, Climatic Change, 122, 387-400, https://doi.org/10.1007/s10584-013-0905-2, 2014.
Pedde, S., Kok, K., Onigkeit, J., Brown, C., Holman, I., and Harrison, P. A.: Bridging uncertainty concepts across narratives and simulations in environmental scenarios, Regional Environmental Change, 19, 655-666, https://doi.org/10.1007/s10113-018-1338-2, 2019.
Py4J: Py4J - A Bridge between Python and Java, https://www.py4j.org/, 2025.
van Vuuren, D. P., Edmonds, J., Kainuma, M., Riahi, K., Thomson, A., Hibbard, K., Hurtt, G. C., Kram, T., Krey, V., Lamarque, J.-F., Masui, T., Meinshausen, M., Nakicenovic, N., Smith, S. J., and Rose, S. K.: The representative concentration pathways: an overview, Climatic Change, 109, 5, https://doi.org/10.1007/s10584-011-0148-z, 2011.
Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., and Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models, Advances in neural information processing systems, 35, 24824-24837, 2022.
Zeng, Y.: LlmInstitution_CRAFTY_data, Zenodo, https://doi.org/10.5281/zenodo.14622334, 2025a.
Zeng, Y.: LlmInstitution_CRAFTY (v1.0), Zenodo, https://doi.org/10.5281/zenodo.14622039, 2025b.