SOCIETY OF PROGRAMS: COMPILING LLM BEHAVIOR INTO SCALABLE SOCIAL DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Imagine a social simulation in which the same kind of decision is made thousands of times: by different agents, under recurring contexts, and across successive timesteps. Asking an LLM to recompute each decision independently makes population-scale simulation expensive even when much of the induced behavior is reusable. We present **Society of Programs (SOP)**, a hierarchical compiler that turns reference-LLM behavior into reusable social dynamics. SOP first compiles task-relevant behavior into executable prototype programs, then groups agents by the responses of these frozen programs and propagates their states through region-level transition operators, and finally merges consecutive updates whenever the propagation conditions remain unchanged. This changes the computational unit from repeated agent-level inference to reusable computation over programs, populations, and temporal blocks. Across WVS and Reddit social simulations, SOP preserves high fidelity to the Full–LLM reference while reducing redundant inference and propagation. In a multi-round opinion simulation with $100{,}000$ agents, it reduces LLM calls by approximately $5{,}163\times$ and build-plus-deployment time by approximately 94% relative to Full–LLM. The gains arise from behavioral regularities that can be compiled and reused; temporal compression is applied only across stable intervals and required output boundaries.

1 INTRODUCTION

Imagine a society of agents making decisions over time. At every round, each agent receives an individual profile, a changing external context, and a state carried forward from earlier decisions. Large language models make this setting expressive: the same model can condition behavior on attributes, history, and context rather than following one fixed rule (Park et al., 2023; Gao et al., 2023). But the same flexibility creates a computational paradox. As the population and simulation horizon grow, the system repeatedly asks the LLM to recompute behavior that may already have appeared in other agents or at earlier timesteps.

This repeated computation is becoming a central obstacle for large-scale LLM social simulation. Social simulation offers a controllable and repeatable way to study collective behavior and support social-science analysis (Gao et al., 2024; Mou et al., 2026), and recent systems have moved from small interaction settings toward larger populations and social dynamics (Park et al., 2024; Yang et al., 2024; Piao et al., 2026). Yet inference volume still grows with the number of agent–timestep decisions, making latency and monetary cost difficult to control (Yang et al., 2024; Guan et al., 2025; Zheng et al., 2026).

Existing work attacks this problem at several layers. Quantization, speculative decoding, KV-cache management, and high-throughput serving make individual calls cheaper or faster (Frantar et al., 2023; Leviathan et al., 2023; Kwon et al., 2023; Sheng et al., 2023). Model cascades, routing, and surrogates reduce the use of expensive models (Chen et al., 2024; Ong et al., 2025; Guan et al., 2025). Archetypes, representatives, and topology-based units reuse decisions across similar agents (Chopra et al., 2025; Zheng et al., 2026; Xu et al., 2026). These approaches are valuable, but the basic computational object usually remains an agent-level decision executed repeatedly as the simulation unfolds. The population and the timeline therefore continue to accumulate redundant work.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We take a different view: an LLM-driven social simulation can be treated as a compilation problem. The reference LLM induces a collection of decision patterns. Some patterns can be expressed as executable programs; similar program responses can be represented by shared population transitions; and stable transitions can be advanced in temporal blocks. This leads to our central question: *Can a society of LLM-induced behaviors be compiled once and reused across agents and time, while remaining faithful to the Full–LLM dynamics?*

We answer this question with **Society of Programs (SOP)**, a hierarchical compilation framework. SOP turns LLM behavior into a society of reusable computational objects through three stages: (1) **Behavior compilation** identifies task-relevant prototypes and learns an executable decision program for each one. Programs are selected using both one-step agreement and closed-loop rollout behavior, so their own state updates are part of the fitting process. (2) **Region compilation** partitions agents according to the actual response differences of the frozen programs and precomputes region-level transition operators. Online execution then propagates population state rather than running a program separately for every agent. (3) **Event-driven propagation** merges consecutive timesteps into temporal blocks when the relevant context, activation semantics, feedback conditions, and requested outputs remain fixed. The block update is equivalent to stepwise propagation of the same compiled model under the stated validity conditions.

The resulting perspective changes what it means to scale an LLM social simulation. Scaling the population need not imply a proportional increase in online LLM inference once behavior has been compiled and shared. Scaling the horizon can benefit from temporal reuse when the simulation contains stable intervals, while new events and output boundaries remain explicit computational boundaries. The framework is complementary to low-level inference and serving optimizations, but it targets a different source of cost: structural repetition in the simulation itself.

Our experiments on WVS and Reddit show that SOP closely approximates the Full–LLM reference while substantially reducing simulation cost, achieving JSDs of 0.1289 and 0.2658, respectively. In the WVS scaling experiment with $100{,}000$ agents, SOP reduces LLM calls by approximately $5{,}163\times$ and build-plus-deployment time by approximately 94% relative to Full–LLM. Region sharing reduces propagation Updates by over 90% on both benchmarks under the reported query schedules, while temporal merging provides a further 87.5% reduction on Reddit relative to stepwise region propagation. Ablations identify closed-loop fitting as the component whose removal causes the largest fidelity degradation, and experiments with nine compiler LLMs demonstrate that the pipeline is portable across models, although fidelity remains compiler-dependent.

The contribution is therefore not a claim that the reference LLM becomes a validated model of human behavior. Rather, it is a computational reorganization: repeated LLM-induced decisions are compiled into reusable behavioral programs, population transitions, and temporal blocks. This organization preserves the measured Full–LLM behavior while exposing a path toward larger and longer social simulations. The SoP code is available at https://anonymous.4open.science/r/SoP-A661.

2 Related Work

Large language models provide a more flexible alternative to rule-based agent-based simulation, allowing agents to produce context-dependent decisions from individual attributes, memory, social context, and external events. Generative Agents (Park et al., 2023) demonstrated LLM-based simulation of everyday social behavior through memory, reflection, and planning, while S^3 (Gao et al., 2023) used LLM agents to model emotion, attitudes, and interactions in social networks. Subsequent work has expanded from small interaction settings to larger populations and social systems, moving from reproducing individual behavior and local interactions toward population-scale social dynamics (Park et al., 2024; Yang et al., 2024; Piao et al., 2026; Gao et al., 2024; Mou et al., 2026; 2024; Wang et al., 2025; Zhang et al., 2025; Rossetti et al., 2024; Li et al., 2024a; Ye et al., 2026).

As population size and simulation horizon increase, invoking an LLM for every agent at every timestep leads to rapidly growing latency and monetary cost. Although many efficient LLM techniques were not developed specifically for social simulation, they can reduce inference overhead. Model quantization, speculative decoding, KV-cache management, and memory- and batching-oriented optimizations can lower per-call cost or improve serving throughput (Frantar et al., 2023;

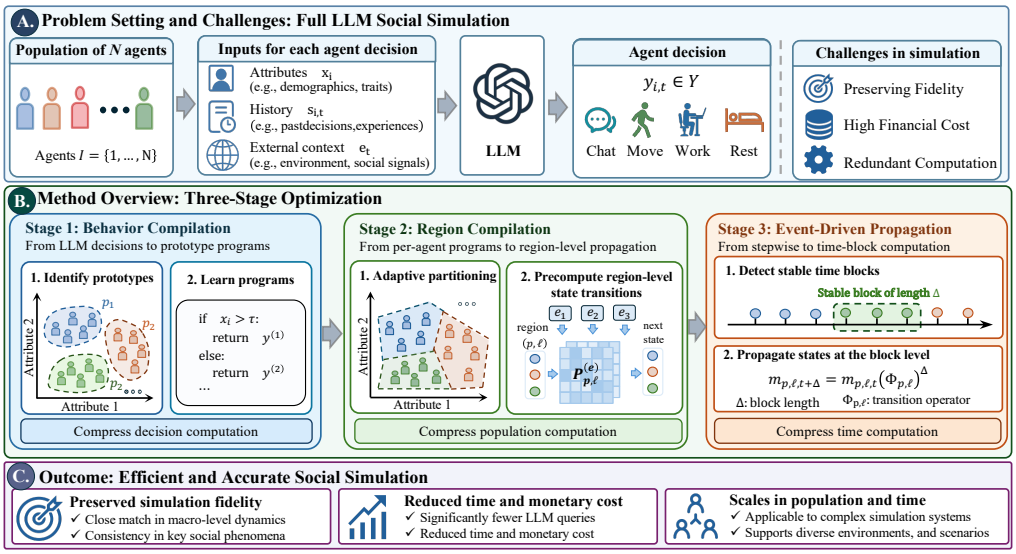


Figure 1: Overview of the Society of Programs (SOP). LLM-induced behavior is compiled into prototype programs, shared through response-aware population regions, and advanced over stable temporal blocks.

Leviathan et al., 2023; Kwon et al., 2023; Sheng et al., 2023; Xiao et al., 2023; Li et al., 2024b; Liu et al., 2024; Yu et al., 2022; Agrawal et al., 2024; Dao et al., 2022; Dao, 2024), while model cascades and query routing reduce cost by limiting calls to expensive models (Chen et al., 2024; Ong et al., 2025). These techniques improve the low-level execution efficiency of LLM social simulation, but generally do not reduce the number of agent–timestep decisions that the simulation itself must process.

Recent work has therefore begun to exploit redundancy in population structure and simulation dynamics directly. LLM Archetypes (Chopra et al., 2025) maps agents with identical or similar feature combinations to a limited set of archetypes, queries the LLM only at the archetype level, and reuses the resulting behavioral distributions for the corresponding individuals. TopoSim (Xu et al., 2026) uses structural similarity in social networks to organize coordination units and reduce repeated decisions. Systems such as Light Society (Guan et al., 2025), OASIS (Yang et al., 2024), and AgentSociety (Piao et al., 2026) improve scalability through surrogate models, parallel execution, and large-scale simulation infrastructure. Together, these studies show that beyond optimizing individual LLM calls, exploiting repeated structure across agents and simulation processes can also substantially reduce the cost of large-scale social simulation. Additional related work is discussed in Appendix A.

3 METHODOLOGY

Figure 1 summarizes the overall framework. We propose SOP, a three-stage optimization method for large-scale, long-horizon LLM social simulation. The three stages address, in sequence, how to replace LLM decisions, how to avoid per-agent program execution, and how to avoid per-timestep propagation. Operational and computational details are provided in Appendices B and C so that the main text can focus on the overall simulation logic.

3.1 PROBLEM SETUP AND OBJECTIVE

We consider a discrete-time LLM-driven social simulation with N agents, $\mathcal{I} = \{1, \ldots, N\}$, over timesteps $\mathcal{T} = \{1, \ldots, T\}$. Each agent has a discrete decision state $\mathcal{Y} = \{y^{(k)}\}_{k=1}^{M}$, static attributes $\boldsymbol{x}_i \in \mathbb{R}^d$, and observes an external context $\boldsymbol{e}_t \in \mathbb{R}^q$ at timestep t.

To support tasks with memory, accumulated evidence, or other evolving information, we define a finite *decision-relevant dynamic state* $s_{i,t} \in \mathcal{S} = \{s^{(r)}\}_{r=1}^{K}$. It contains the information needed for

future decisions, while $y_{i,t}$ is the decision state reported and evaluated. When no additional dynamic information is needed, we set $\mathcal{S} = \mathcal{Y}$, $K = M$, and $s_{i,t} = y_{i,t}$. We assume that $s_{i,t}$ is sufficient to represent the dynamic information required by subsequent LLM decisions.

Let $\mathcal{A}_t \subseteq \mathcal{I}$ denote the agents that update at timestep t. The reference LLM simulation is

$$y_{i,t} = \begin{cases} \mathcal{O}_{\mathrm{LLM}}(\boldsymbol{x}_i, \boldsymbol{e}_t, s_{i,t-1}), & i \in \mathcal{A}_t, \\ y_{i,t-1}, & i \notin \mathcal{A}_t, \end{cases} \qquad y_{i,t} \in \mathcal{Y}, \tag{1}$$

where $\mathcal{O}_{\mathrm{LLM}}$ denotes the reference LLM decision process and $y_{i,0}$ is the initial decision state. After each decision, the environment updates $s_{i,t}$ using the task-specific rule $\mathcal{U}$ based on the previous dynamic state, the decision outcome, and the current context. Thus, $y_{i,t}$ is the output of interest, whereas $s_{i,t}$ carries forward the information needed for later decisions. Repeated LLM inference across agents and timesteps becomes increasingly costly as N and T grow.

Our goal is to reduce this cost while preserving the reference LLM's decision patterns. Behavior compilation replaces agent-level LLM decisions with executable programs, while region compilation and event-driven propagation further exploit redundancy across agents and time. We evaluate fidelity through the aggregate decision-state distribution

$$\pi_t = \left(\pi_t^{(1)}, \ldots, \pi_t^{(M)}\right), \qquad \pi_t^{(k)} = \frac{1}{N} \sum_{i \in \mathcal{I}} \mathbb{I}\left[y_{i,t} = y^{(k)}\right], \tag{2}$$

where $\mathbb{I}[\cdot]$ is the indicator function and $\sum_{k=1}^{M} \pi_t^{(k)} = 1$. The computational objective is to obtain an accurate approximation $\widehat{\pi}_t$ of π_t while substantially reducing simulation cost.

3.2 Behavior Compilation: From LLM Decisions to Prototype Programs

Behavior compilation replaces online LLM decisions with locally executable programs. To capture behavioral heterogeneity, we identify task-relevant prototypes and learn one decision program for each. The resulting programs concentrate LLM inference in a one-time compilation stage and replace per-decision LLM calls with local evaluations during deployment, independently of temporal merging.

LLMs can search effectively for executable programs when coupled with external evaluation, as demonstrated by FunSearch (Romera-Paredes et al., 2024) and LLM-SR (Shojaee et al., 2025). Following this paradigm, we use the LLM to propose program structures and calibrate their parameters against reference decisions, with the specific goal of compiling agent-level LLM behavior into reusable decision programs. Implementation details are provided in Appendix B.

Task-relevant prototypes. Fixed demographic partitions may overlook attributes that drive decisions in a given task. We instead use controlled LLM decision tests to identify agent attributes that induce larger changes in reference decisions and use them to construct prototypes. Let $\mathcal{P}$ denote the resulting set, where each $p \in \mathcal{P}$ groups agents with similar task-relevant attributes. Construction, scoring, and partitioning are detailed in Appendix B.1.

Prototype decision programs. For each prototype, behavioral regularities extracted from reference decisions guide candidate program generation. For agent i assigned to prototype p, the compiled decision is

$$\widehat{y}_{i,t} = F_p(\widetilde{\boldsymbol{x}}_i, \widetilde{\boldsymbol{e}}_t, s_{i,t-1}; \boldsymbol{\theta}_p), \qquad \widehat{y}_{i,t} \in \mathcal{Y}, \tag{3}$$

where $\widetilde{\boldsymbol{x}}_i$ and $\widetilde{\boldsymbol{e}}_t$ are normalized agent and context features, $s_{i,t-1}$ is the dynamic state, and $\boldsymbol{\theta}_p$ denotes program parameters.

Candidate programs are calibrated against reference decisions and selected using one-step fidelity, closed-loop rollout fidelity, and program complexity. Closed-loop evaluation accounts for error accumulation by updating $s_{i,t}$ with the same task-specific rule $\mathcal{U}$ used in the reference simulation, so selection reflects recursive execution rather than isolated predictions. Rule extraction, program representation, grammar constraints, parameter optimization, and structural search are detailed in Appendices B.2–B.4.

After selection, prototype assignments and programs are frozen. Online simulation evaluates F_p locally without further LLM calls, and region compilation subsequently reduces the remaining per-agent evaluations.

3.3 Region Compilation: From Per-Agent Programs to Population State Propagation

Behavior compilation removes expensive LLM calls, but evaluating the compiled program for every agent at every timestep can still be redundant at scale. Region compilation exploits behavioral similarity across agents to replace per-agent execution with population-level state propagation.

For each prototype $p \in \mathcal{P}$, we adaptively partition agents according to response differences of the frozen program F_p, producing regions $\{\mathcal{R}_{p,\ell}\}_{\ell \in \mathcal{L}_p}$. Agents remain in the same region when their program responses are similar across external contexts and dynamic states, while regions with larger response variation are split further. Construction and stopping criteria are detailed in Appendix C.1.

Once the regions are fixed, we execute the frozen programs together with the task-specific state-update rule offline to precompute region-level transitions under different external contexts. Let $\mathcal{S}_{p,\ell}^{\mathrm{cal}}$ denote the calibration agents in region (p, ℓ). For an external-context vector $\boldsymbol{e}$ with normalized representation $\widetilde{\boldsymbol{e}}$, we define

$$P_{p,\ell}^{(\boldsymbol{e})}[r, r'] = \frac{1}{|\mathcal{S}_{p,\ell}^{\mathrm{cal}}|} \sum_{i \in \mathcal{S}_{p,\ell}^{\mathrm{cal}}} \mathbb{I}\Big[\mathcal{U}\Big(s^{(r)}, F_p\Big(\widetilde{\boldsymbol{x}}_i, \widetilde{\boldsymbol{e}}, s^{(r)}; \boldsymbol{\theta}_p\Big), \boldsymbol{e}\Big) = s^{(r')}\Big], \tag{4}$$

where $r, r' \in \{1, \ldots, K\}$ index the dynamic states before and after the update. Thus, $P_{p,\ell}^{(\boldsymbol{e})}[r, r']$ is the empirical transition probability from $s^{(r)}$ to $s^{(r')}$ within region (p, ℓ) under context $\boldsymbol{e}$.

The shared transition matrix averages program-induced transitions over calibration agents and therefore approximates the dynamics obtained from per-agent execution. Exactness conditions and sources of aggregation error are analyzed in Appendix C.6. When the dynamic state coincides with the current decision, $\mathcal{S} = \mathcal{Y}$ and Eq. (4) operates directly over decision states; otherwise, propagation uses the corresponding extended state space $\mathcal{S}$.

After compilation, online simulation propagates region-level state distributions instead of evaluating F_p separately for each agent. These states are mapped back to the M decision categories in Eq. (2) to recover the aggregate decision-state distribution. Further details on region representation and update rules are provided in Appendix C.

3.4 Event-Driven Propagation: From Per-Timestep Updates to Temporal Blocks

Region compilation removes per-agent program execution, but propagation can still be redundant when consecutive timesteps share the same region-level dynamics. Event-driven propagation addresses this temporal redundancy by merging such timesteps into blocks and starting a new block only when the propagation conditions change.

We partition the timeline according to changes in external context or execution conditions. Within each block, the propagation dynamics of every region remain fixed. A new block begins when an external event, execution mechanism, feedback state, or requested output changes. Detailed boundary rules are given in Appendix C.3.

Let $\boldsymbol{m}_{p,\ell,t} \in \mathbb{R}_{\geq 0}^{K}$ denote the dynamic-state population counts of region $\mathcal{R}_{p,\ell}$ at timestep t. Suppose a block starting at t spans Δ timesteps and uses a fixed one-step propagation operator $\Phi_{p,\ell}$. When linear propagation applies, the block endpoint is

$$\boldsymbol{m}_{p,\ell,t+\Delta} = \boldsymbol{m}_{p,\ell,t} \left(\Phi_{p,\ell}\right)^{\Delta}. \tag{5}$$

Thus, repeated intermediate updates need not be executed explicitly when the propagation dynamics remain unchanged.

The exact block update depends on the activation semantics. With independent reactivation, powers of the fixed one-step operator can be applied directly. Under without-replacement activation, the activation mass is first aggregated over the block and the endpoint is computed in one update. In both cases, computation is merged while propagation conditions remain stable and the timeline is split whenever they change. Detailed formulas, validity conditions, and feedback handling are provided in Appendices C.4 and C.5. Further analysis of approximation error and computational complexity is provided in Appendix C.6.

4 EXPERIMENTS

This section first describes the experimental setup and then addresses the following research questions:

RQ1: How faithfully can SOP reproduce the population-level and individual-state dynamics of a Full–LLM simulation?

RQ2: How does cost scale with population size and simulation horizon, and when does a single run amortize the compilation cost?

RQ3: How do behavior-compilation and propagation components contribute to fidelity and computational savings?

RQ4: How sensitive is the framework to the choice of compiler LLM?

4.1 EXPERIMENTAL SETUP

Tasks and datasets. We use two complementary benchmarks. The WVS benchmark follows the population attributes and multi-round public-event setting constructed from World Values Survey (WVS) Wave 7[1] in APS (Zheng et al., 2026), capturing heterogeneous population responses to changing external contexts. The Reddit benchmark is constructed from the user profiles and social-interaction data of OASIS (Yang et al., 2024), and provides longer-horizon dynamics with sparse activation and evolving events. Together, they cover structured short-horizon population responses and long-horizon social-platform dynamics. Further construction and scenario details follow APS (Zheng et al., 2026) and OASIS (Yang et al., 2024).

Full–LLM reference. For each benchmark, we independently generate the Full–LLM simulation defined in Section 3.1. The resulting trajectory is held fixed and used only as the reference against which approximation methods are evaluated.

Methods and operating point. We compare three representative cost-reduction strategies. Attribute Archetype, adapted from LLM Archetypes (Chopra et al., 2025), groups agents by similar attributes. Coord-Representative, adapted from TopoSim (Xu et al., 2026), forms coordination units using social structure and agent similarity and queries representative agents. LS-Surrogate follows the model-substitution approach of Light Society (Guan et al., 2025) and trains a lightweight predictor on reference decisions. Unless otherwise stated, we use *deepseek/deepseek-v4-flash-0731* through OpenRouter as the primary LLM. Implementation details and hyperparameters are provided in the experimental appendix.

Evaluation metrics. Behavioral fidelity is measured by the mean Jensen–Shannon divergence (JSD) between predicted and Full–LLM population distributions over time, and by Exact Match at the individual-state level. Lower JSD and higher Exact Match indicate better agreement. For efficiency, we report LLM Calls, token usage, estimated LLM inference cost, and wall-clock time, accounting for both offline construction and online execution. For SOP, offline cost mainly arises from behavior and region compilation, while online cost is dominated by state propagation.

Fair comparison protocol. We account for both offline and online costs and, where possible, use the same LLM backend, prompts, and inference settings across methods. Detailed cost accounting, baseline adaptations, and system settings are provided in Appendix F.

4.2 BEHAVIORAL FIDELITY (RQ1)

Table 1 summarizes the time-averaged JSD and Exact Match on WVS and Reddit, measuring fidelity to the Full–LLM reference at the population-distribution and individual-state levels, respectively. Overall, SOP achieves the lowest mean JSD and the highest Exact Match in both settings. On WVS, its JSD is 0.1289, a 43.2% relative reduction from the closest baseline, Attribute Archetype. Exact Match increases from 0.3593 to 0.6702, an improvement of 31.09 percentage points, indicating that SOP better preserves both population distributions and individual states across repeated decision rounds.

[1] https://www.worldvaluessurvey.org/WVSDocumentationWV7.jsp

Table 1: Behavioral fidelity on WVS and Reddit. For SOP, per-agent closed-loop program replay uses the same three compilations as the ablation study, with metrics averaged across runs. Point estimates and conditional 95% bootstrap confidence intervals are reported separately. Lower JSD and higher Exact Match are better; bold marks the best point estimate in each setting.

WVS				
Method	JSD ↓	JSD CI	Exact Match ↑	Exact Match CI
LS-Surrogate	0.3390	[0.3159, 0.3651]	0.2814	[0.2594, 0.3039]
Attribute Archetype	0.2270	[0.2076, 0.2494]	0.3593	[0.3349, 0.3840]
Coord-Representative	0.4042	[0.3831, 0.4279]	0.2296	[0.2114, 0.2481]
SOP	**0.1289**	[0.1174, 0.1420]	**0.6702**	[0.6530, 0.6870]
Reddit				
Method	JSD ↓	JSD CI	Exact Match ↑	Exact Match CI
LS-Surrogate	0.2833	[0.2376, 0.3506]	0.2528	[0.1932, 0.3157]
Attribute Archetype	0.4012	[0.3442, 0.4650]	0.2346	[0.1750, 0.3032]
Coord-Representative	0.2899	[0.2347, 0.3562]	0.3297	[0.2559, 0.4083]
SOP	**0.2658**	[0.1844, 0.3603]	**0.4808**	[0.3580, 0.5968]

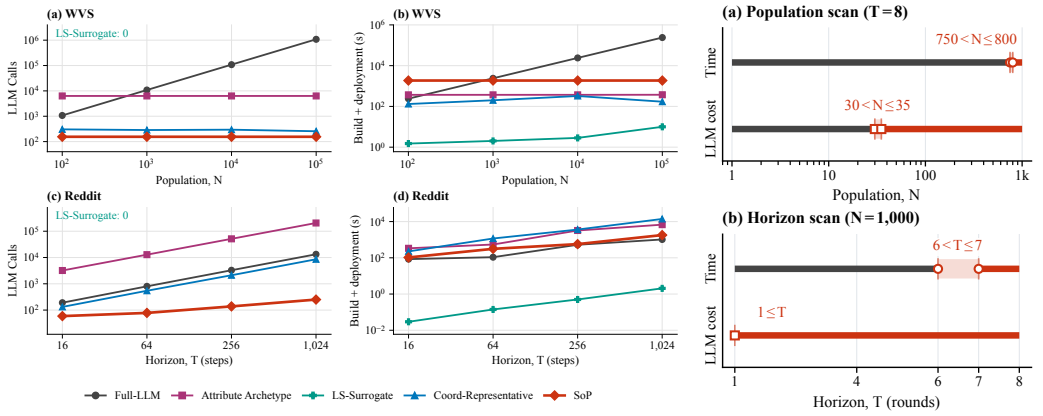


Figure 2: Computational scaling and single-run break-even. Left: Method-specific cost across population sizes and simulation horizons. Right: Empirical break-even intervals at which SOP becomes cheaper or faster than Full–LLM when the full compilation cost is included.

On Reddit, SOP again obtains the best point estimates, with a JSD of 0.2658, below LS-Surrogate, Attribute Archetype, and Coord-Representative. Relative to the closest JSD baseline, LS-Surrogate, this is a reduction of about 6.2%. Exact Match reaches 0.4808, 15.11 percentage points above the strongest baseline, Coord-Representative. Across the two settings, SOP therefore provides a stable approximation to Full–LLM behavioral dynamics and establishes the fidelity basis for the computational-efficiency comparisons that follow.

4.3 Cost and Scalability Across Simulation Sizes (RQ2)

To reflect the two benchmark settings, we scale population size on WVS with a fixed horizon ($T = 8$), and extend the horizon on Reddit with a fixed population ($N = 100$). Cost includes each method's one-time construction and online execution. Figure 2 summarizes both scaling behavior and single-run break-even points.

Scaling behavior. On WVS, Figure 2 (left) shows that SOP reuses the same frozen compilation as N increases. Its LLM Calls remain nearly constant and runtime grows only slightly, whereas Full–LLM requires per-agent inference and scales rapidly with population size. Additional agents therefore mainly increase local state maintenance and propagation rather than LLM inference.

On Reddit, LLM Calls and runtime increase with T because longer horizons introduce additional events, state transitions, and decision contexts. The gain depends on reuse: recurring contexts amortize compilation across timesteps, whereas frequent new states or decision contexts require additional inference.

Table 2: Ablation results on WVS and Reddit. JSD and Exact Match are mean ± s.d. over three compilations; Updates are mean propagation counts.

Variant	WVS		Reddit	
A. Program Replay	JSD ↓	Exact Match (%) ↑	JSD ↓	Exact Match (%) ↑
SOP	0.1289 ± 0.0584	67.02 ± 2.94	0.2658 ± 0.0155	48.08 ± 0.36
A1 Preset Attributes	0.1694 ± 0.0222	65.19 ± 2.49	0.2875 ± 0.0051	45.72 ± 2.46
A2 No Rule Guidance	0.1514 ± 0.0199	65.05 ± 3.03	0.2855 ± 0.0222	47.99 ± 1.00
A3 No Closed-Loop Fit	0.1796 ± 0.0283	51.52 ± 10.81	0.3076 ± 0.0164	42.89 ± 6.34
B. Propagation	JSD ↓	Updates ↓	JSD ↓	Updates ↓
SOP	0.1295 ± 0.0589	147	0.2686 ± 0.0143	1,024
A4 No Region Sharing	0.1295 ± 0.0589	8,000	0.2686 ± 0.0143	12,800
A5 No Temporal Merge	0.1295 ± 0.0589	147	0.2686 ± 0.0143	8,192

Note: Updates use every-round WVS queries and 8-step block-end Reddit queries; Reddit Update counts are measured separately from JSD evaluation.

Table 3: Robustness across compiler LLMs under fixed reference trajectories.

Compiler Model	WVS		Reddit	
	JSD ↓	Exact Match (%) ↑	JSD ↓	Exact Match (%) ↑
deepseek/deepseek-v4.1-flash	0.1148 ± 0.0370	63.96 ± 4.66	0.1280 ± 0.0335	32.99 ± 1.86
openai/gpt-6-astra	0.0761 ± 0.0096	62.29 ± 0.47	0.0994 ± 0.0359	35.70 ± 3.46
google/gemini-3.8-flash	0.0812 ± 0.0341	62.58 ± 2.75	0.1560 ± 0.0116	31.52 ± 2.08
tencent/hy4-preview	0.0780 ± 0.0077	64.38 ± 2.54	0.1641 ± 0.1045	32.69 ± 1.37
z-ai/glm-5.3-flash	0.0682 ± 0.0254	64.42 ± 1.21	0.1665 ± 0.0578	26.44 ± 2.99
qwen/qwen3.8-flash	0.0686 ± 0.0059	58.17 ± 3.34	0.1828 ± 0.0966	29.16 ± 2.08
moonshotai/kimi-k3	0.0645 ± 0.0228	62.50 ± 0.75	0.1331 ± 0.0282	30.58 ± 2.40
minimax/minimax-m3	0.0563 ± 0.0032	61.54 ± 2.09	0.2126 ± 0.0247	28.38 ± 1.29
xiaomi/mimo-v2.5	0.0879 ± 0.0340	64.42 ± 3.18	0.1989 ± 0.0459	30.23 ± 6.06

Note: JSD evaluates region-level expected propagation. Exact Match measures the fraction of per-agent program-replay outputs that exactly match the corresponding Full–LLM reference states. WVS evaluates all agent–round pairs, while Reddit evaluates only actually activated decisions. Full metrics and confidence intervals are reported in Appendix E.3.

Single-run break-even. Scaling trends alone do not show when a single run amortizes the one-time compilation cost. We therefore perform two WVS scans, varying population size at $T = 8$ and horizon at $N = 1{,}000$, while including the full compilation cost in every run. Figure 2 (right) shows a clear separation between monetary and runtime break-even: LLM cost becomes lower at much smaller scales, whereas wall-clock savings require more computation to amortize compilation overhead. For example, at $T = 8$, the cost break-even occurs at $30 < N \leq 35$, compared with $750 < N \leq 800$ for runtime; the horizon scan shows the same pattern, with cost savings from $T = 1$ but runtime savings only after six rounds. Thus, reducing paid inference and amortizing compilation latency are distinct deployment considerations.

Overall, behavior compilation reduces online LLM inference, while region sharing reduces repeated local computation across agents. These benefits remain available even when outputs are required at every timestep; temporal merging provides an additional reduction only when stable intervals permit multiple steps between required outputs or interventions.

4.4 Ablation Study (RQ3)

We ablate behavior compilation and propagation separately. A1–A3 replace task-relevant partitioning, remove rule guidance, and remove closed-loop fitting, while A4–A5 disable region sharing and temporal merging. Table 2 summarizes the results; further details are provided in Appendix D.

For behavior compilation, the full method achieves the best fidelity on both benchmarks. Removing closed-loop fitting (A3) causes the largest degradation in JSD and Exact Match, indicating that one-step agreement alone is insufficient when prediction errors affect subsequent states. A1 and A2 yield smaller losses, suggesting that task-relevant partitioning and rule guidance mainly improve the representation of behavioral heterogeneity and constrain program search.

The propagation ablations isolate reuse across agents and time under the same query schedule. Region sharing reduces Updates by approximately 98.2% on WVS and 92.0% on Reddit relative to

A4, showing substantial population-level redundancy even without temporal compression. On Reddit, temporal merging further reduces Updates by 87.5% relative to A5 through block-level propagation, whereas WVS gains no additional temporal reduction because every round introduces an event change and query boundary. Overall, the ablations show that closed-loop behavior compilation improves fidelity, while region sharing and temporal merging exploit population and temporal redundancy, respectively.

4.5 Robustness to the Compiler LLM (RQ4)

To test whether behavior compilation depends on a particular LLM, we replace the compiler model used for prototype extraction, rule induction, and program generation with each of the nine LLMs in Table 3, then rerun region compilation and propagation using the newly generated programs. Within each dataset, all models use the same data range, prompt templates, and search-budget rules, and each is compiled independently three times. Explicit reasoning is disabled by default; GPT-6 Astra, Gemini 3.8 Flash, and GLM 5.3 Flash use the low reasoning setting required by their interfaces. Because these evaluation subsets differ from the main experiment, the results in this section are not directly compared with the main table. Full details appear in Appendix E.

All nine models successfully complete behavior compilation and support subsequent propagation, indicating that the framework does not depend on a single compiler LLM. The generated programs remain usable for region propagation and temporal merging across models; numerical-equivalence checks are reported in the appendix. WVS JSD ranges from 0.0563 to 0.0879, while Reddit ranges from 0.0994 to 0.2126. Relative performance is not consistent across tasks: for example, MiniMax M3 achieves the lowest JSD on WVS, whereas GPT-6 Astra achieves both the lowest JSD and the highest Exact Match on Reddit. The framework is therefore portable across compiler models, although the choice of compiler still affects final behavioral fidelity.

5 Limitations

This work focuses on compiling online LLM decisions into reusable behavioral programs to reduce repeated inference in population-scale simulation, but several limitations remain. First, our fidelity metrics measure agreement between compiled programs or propagated results and the Full–LLM reference trajectory; they do not establish that the reference LLM itself accurately models real human behavior. Any systematic bias in the reference model may therefore be preserved by the compilation process. Second, the computational gains depend on behavioral regularities being reusable across agents and time. If the environment continually introduces new events, state combinations, or decision patterns, coverage by existing programs may fall, increasing new LLM calls and reducing the benefits of spatial sharing and temporal merging. Region-level probabilistic propagation may also smooth fine-grained individual differences and dynamic effects arising from complex interactions. Future work should further evaluate behavioral validity and generalization across additional domains, reference models, richer interaction networks, and real human behavioral data.

6 Discussion and Conclusion

Our central observation is that the computational redundancy of large-scale LLM social simulation comes not only from individual model calls, but also from repeatedly executing similar decisions across agents and timesteps. Society of Programs (SoP) treats this redundancy as a compilation opportunity: it restructures per-agent, per-timestep simulation into three reusable layers—behavioral programs, region-level propagation, and temporal blocks—reducing redundancy across the decision, population, and temporal dimensions. Experiments on WVS and Reddit show that this design maintains strong behavioral fidelity while improving scalability with population size and simulation horizon. More broadly, our results show that scalability can come not only from faster serving or cheaper models, but also from exploiting structural repetition in LLM-induced behavior. This offers a path toward turning expensive online simulation into a reusable and scalable population-level computation.

AI USE STATEMENT

Generative AI tools were used to assist with language translation, grammar and style editing, and LaTeX formatting. They were also used to support limited literature discovery, wording refinement, and coding assistance. All AI-assisted content was reviewed and revised by the authors, and all cited references were manually checked against the original sources. Generative AI was not used to generate experimental data or results, nor to make final scientific judgments.

ETHICS AND SOCIAL IMPACT

Our method is designed to reduce the computational cost of LLM-based social simulation. Its approximation target is the Full–LLM reference induced by a specified model, prompt, population profile, external events, and simulation mechanism, not real human behavior. The reported fidelity therefore measures how closely the approximation reproduces the state distribution and individual decisions of the Full–LLM simulation; it should not be interpreted as a direct measurement of public opinion, policy preference, or real-world social behavior. LLM decisions may themselves reflect biases in training data, prompt design, model versions, or input profiles, and our method may inherit these biases. Consequently, low JSD or high Exact Match against the Full–LLM reference does not imply comparable validity with respect to real populations. Applications that make claims about real social behavior still require independent human data, domain expertise, and appropriate empirical validation.

Behavior compilation, region sharing, and temporal reuse also introduce risks specific to approximation. Mapping similar individuals or states to reusable programs and propagation units can substantially reduce repeated inference, but high aggregate fidelity does not guarantee equally accurate preservation of minority groups, rare states, or atypical behaviors. Analyses involving vulnerable groups or other high-impact settings should therefore examine subgroup errors and tail behavior and explicitly disclose cases that cannot be modeled reliably. Improved computational efficiency also creates dual-use risk: the same capability can support social-science research, policy stress testing, and scenario analysis, but may also lower the technical barrier to large-scale profiling, opinion manipulation, political persuasion, or microtargeting. We do not recommend using the method for manipulative persuasion or voter targeting, or for high-impact decisions in domains such as employment, credit, healthcare, education, or public safety without independent review and empirical validation.

The use of individual profiles and social-interaction data further requires appropriate data governance and privacy safeguards. Survey coverage, nonresponse, cultural context, platform-selection effects, and profile construction may all affect simulation outcomes; data derived from real users may additionally create risks of privacy loss, re-identification, or unintended secondary use. Practical deployments should follow applicable data-use and privacy requirements and, where possible, rely on de-identified, aggregated, or synthetic data. They should also document the base model and version, prompt templates, profile construction, behavior-compilation configuration, key approximation parameters, and known failure modes, while clearly separating computational fidelity to the Full–LLM reference from empirical validity with respect to human behavior. The contribution of this work is to reduce the cost of an existing LLM social-simulation process, not to improve its social-scientific realism; without external validation, the resulting outputs are better suited to model analysis, method comparison, and exploratory scenario studies than to deterministic claims about real social groups.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *The 18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 117–134, 2024.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

Ayush Chopra, Shashank Kumar, Nurullah Giray Kuru, Ramesh Raskar, and Arnau Quera-Bofarull. On the limits of agency in agent-based models. In *The 24th International Conference on Autonomous Agents and Multiagent Systems*, pp. 500–509, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FLASHATTENTION: Fast and memory-efficient exact attention with IO-awareness. In *The 36th International Conference on Neural Information Processing Syste*, volume 35, pp. 16344–16359, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. S^3: Social-network simulation system with large language model-empowered agents. *arXiv preprint arXiv:2307.14984*, 2023.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, 11(1):1259, 2024.

Haoxiang Guan, Jiyan He, Liyang Fan, Zhenzhen Ren, Shaobin He, Xin Yu, Yuan Chen, Shuxin Zheng, Tie-Yan Liu, and Zhen Liu. Modeling earth-scale human-like societies with one billion agents. *arXiv preprint arXiv:2506.12078*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *The 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *The 40th International Conference on Machine Learning*, volume 202, pp. 19274–19286, 2023.

Nian Li, Chen Gao, Mingyu Li, Yong Li, and Qingmin Liao. EconAgent: Large language model-empowered agents for simulating macroeconomic activities. In *The 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 15523–15536, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024b.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *The 41st International Conference on Machine Learning*, 2024.

Xinyi Mou, Zhongyu Wei, and Xuan-Jing Huang. Unveiling the truth and facilitating change: Towards agent-based large-scale social movement simulation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4789–4809, 2024.

Xinyi Mou, Xuanwen Ding, Qi He, Liang Wang, Jingcong Liang, Xinnong Zhang, Libo Sun, Jiayu Lin, Jie Zhou, Xuanjing Huang, and Zhongyu Wei. From individual to society: A survey on social simulation driven by Large Language Model-Based agents. *ACM Computing Surveys*, 58 (11):1–41, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. In *The Thirteenth International Conference on Learning Representations*, 2025.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *The 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Joon Sung Park, Carolyn Q. Zou, Aaron Shaw, Benjamin Mako Hill, Carrie Cai, Meredith Ringel Morris, Robb Willer, Percy Liang, and Michael S. Bernstein. Generative agent simulations of 1,000 people. *arXiv preprint arXiv:2411.10109*, 2024.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, Chen Gao, Fengli Xu, Fang Zhang, Ke Rong, Jun Su, and Yong Li. AgentSociety: Large-scale simulation of LLM-driven generative agents advances understanding of human behaviors and society. *iFuture*, 2026. doi: 10.26599/IF.2026.9710004.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Giulio Rossetti, Massimo Stella, Rémy Cazabet, Katherine Abramski, Erica Cau, Salvatore Citraro, Andrea Failla, Riccardo Improta, Virginia Morini, and Valentina Pansanella. Y Social: an LLM-powered social media digital twin. *arXiv preprint arXiv:2408.00818*, 2024.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *The 40th International Conference on Machine Learning*, volume 202, pp. 31094–31116, 2023.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jiakai Tang, Heyang Gao, Xuchen Pan, Lei Wang, Haoran Tan, Dawei Gao, Yushuo Chen, Xu Chen, Yankai Lin, Yaliang Li, Bolin Ding, Jingren Zhou, Jun Wang, and Ji-Rong Wen. GenSim: A general social simulation platform with large language model based agents. In *The 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*, pp. 143–150, 2025.

Lei Wang, Heyang Gao, Xiaohe Bo, Xu Chen, and Ji-Rong Wen. YuLan-OneSim: Towards the next generation of social simulator with large language models. In *Workshop on Scaling Environments for Agents*, 2025.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *The 40th International Conference on Machine Learning*, pp. 38087–38099, 2023.

Yuwei Xu, Shulun Zhang, Yingli Zhou, Shipei Zeng, Laks V. S. Lakshmanan, and Chenhao Ma. Topology-aware LLM-driven social simulation: A unified framework for efficient and realistic agent dynamics. *arXiv preprint arXiv:2604.18011*, 2026.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, Prateek Gupta, Shuyue Hu, Zhenfei Yin, Guohao Li, Xu Jia, Lijun Wang, Bernard Ghanem, Huchuan Lu, Chaochao Lu, Wanli Ouyang, Yu Qiao, Philip Torr, and Jing Shao. OASIS: Open agent social interaction simulations with one million agents. *arXiv preprint arXiv:2411.11581*, 2024.

Yulei Ye, Wenhao Li, Zhong Wen, Yunshu Huang, Yichen Hu, Zifan Wei, Yige Wang, Xinyu Xie, Haoxuan Yang, Yanjun Huang, Ruijia Li, Hong Qian, Yu Song, Bo Jiang, Bingdong Li, Lijun Li, Bo Zhang, Pinlong Cai, Xingcheng Xu, Shuangye Chen, Xia Hu, Liang He, Aimin Zhou, Jingjing Qu, Jing Shao, and Xiangfeng Wang. Agentschool: An llm-powered multi-agent simulation for education. *ArXiv*, abs/2605.30144, 2026.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *The 16th USENIX Symposium on Operating Systems Design and Implementation*, pp. 521–538, 2022.

Xinnong Zhang, Jiayu Lin, Xinyi Mou, Shiyue Yang, Xiawei Liu, Libo Sun, Hanjia Lyu, Yihang Yang, Weihong Qi, Yue Chen, et al. SocioVerse: A world model for social simulation powered by LLM agents and a pool of 10 million real-world users. *arXiv preprint arXiv:2504.10157*, 2025.

Quan Zheng, Yan Gao, Shaobin He, Haoxiang Guan, Yuanhe Tian, Jie Feng, Ming Wang, Shuxin Zheng, and Zhen Liu. APS: Bias-controlled adaptive prototype simulation for population-scale LLM agents. *arXiv preprint arXiv:2605.27419*, 2026.

A Additional Related Work

This section supplements Section 2 with work related to the computational efficiency of large-scale LLM social simulation. Because efficient LLM techniques and large-scale agent systems are not always designed specifically to accelerate social simulation, we organize them by the type of computational cost they reduce and discuss how they relate to large-scale simulation.

A.1 LLM Inference and Serving Optimization

When many LLM agents run concurrently, the most direct cost comes from model inference itself, so general LLM inference optimizations can also reduce the low-level execution cost of social simulation. Model quantization reduces memory and compute requirements by lowering parameter precision, as in GPTQ (Frantar et al., 2023). Speculative decoding uses a smaller model to propose candidate tokens that are verified in parallel by the target model, reducing the serial overhead of autoregressive generation (Leviathan et al., 2023). PagedAttention/vLLM (Kwon et al., 2023) improves memory utilization and serving throughput through more efficient KV-cache management, while FlexGen (Sheng et al., 2023) schedules model parameters and intermediate states across GPU, CPU, and storage to improve throughput under constrained hardware. Similar batching, cache-management, and asynchronous-execution techniques can be directly reused by LLM-agent systems. These methods share a common property: they make each LLM call faster or cheaper while largely preserving the original agent decision process. They can therefore improve system throughput substantially, but if each agent must still make an independent decision at every timestep, total computation continues to grow with population size and simulation horizon.

A.2 Model Routing, Substitution, and Surrogates

A second approach reduces the number of requests handled by expensive models. FrugalGPT (Chen et al., 2024) uses model cascades to select among models with different costs and capabilities, while RouteLLM (Ong et al., 2025) learns routing policies that assign queries between high-capability and low-cost models. The same idea naturally extends to LLM-agent simulation: a high-capability model can generate a subset of reference decisions, after which distillation, supervised learning, or other surrogate models approximate its behavior so that most online decisions are produced more cheaply. Large-scale social-simulation systems such as Light Society (Guan et al., 2025) also combine lightweight surrogates, routing, and precomputation to reduce the frequency of expensive model usage. These approaches primarily compress the cost of the *decision model* itself, but the substitute model may still need to run separately for many agents and timesteps. Redundancy across population and time therefore does not necessarily disappear, and changes in the reference LLM, task context, or agent distribution may require new data collection or recalibration.

A.3 Large-Scale LLM Social-Simulation Systems

As the number of LLM agents grows, research has also improved scalability at the simulation-framework and systems level. OASIS (Yang et al., 2024) supports interaction and information diffusion among large populations in social-media environments. AgentSociety (Piao et al., 2026) provides infrastructure for large-scale agent–environment interaction and social experiments, while GenSim (Tang et al., 2025) organizes diverse LLM agents and social scenarios within a general social-simulation platform. Light Society (Guan et al., 2025) further introduces structured state representations, batched computation, and event-driven execution for very large social processes. These systems typically combine parallelism, asynchronous scheduling, batching, caching, modular environments, and event management to support more agents with limited resources. Compared with optimizing model inference alone, they broaden optimization to the simulation runtime as a whole. Their primary objective, however, is generally better scheduling and resource utilization rather than redefining the fundamental computational unit of individual decision making and population-state propagation.

A.4 Representative Agents, Archetypes, and Population Propagation

A more direct approach to population scaling is to exploit similarity across agents so that decisions from a subset of representatives can serve larger groups. On the Limits of Agency in Agent-based Models (Chopra et al., 2025) represents populations with a finite set of LLM archetypes defined by identical or similar feature combinations; it obtains behavioral distributions at the archetype level and reuses them for corresponding individuals, avoiding a full LLM decision for every agent. TopoSim (Xu et al., 2026) forms coordination units using similarities in social-network topology, structural roles, and interaction context, allowing structurally similar agents to share representative updates. Unlike methods that only improve serving throughput, these approaches directly reduce the number of agents that require independent expensive decisions and can therefore exploit population-level redundancy. However, existing methods usually continue to query representative LLM agents online or repeat representative decision making and propagation each round, so their cost still depends on the simulation horizon and online query budget.

A.5 Relation to Our Method

The methods above improve the computational efficiency of large-scale LLM-agent simulation at different layers: model inference, model selection, system execution, and population representation. Our work explores a complementary possibility. Once the decision patterns of the reference LLM can be compiled offline, online simulation can move beyond executing a decision model for every agent at every timestep. We first compile reference LLM behavior into executable programs for task-relevant prototypes, then use similarity in program responses to construct population-level behavioral regions, and finally exploit temporal stability in propagation to perform block-level computation. The three stages therefore reduce redundant computation introduced by the decision model, population scale, and simulation horizon, and are complementary to the inference and system optimizations discussed above.

B Implementation Details of Behavior Compilation

This appendix provides reproducible implementation details for the behavior-compilation stage in Section 3.2. It covers construction and screening of task-relevant factor–attribute pairs, controlled pre-decision tests, prototype encoding, sampling and consolidation of natural-language rules, input normalization, decision-state quantization and abstract-syntax-tree constraints, as well as parameter optimization and structural search. The main method uses only the frozen prototype partition and decision programs produced by these procedures; no intermediate step is required for online inference.

B.1 Candidate Factor–Attribute Pairs and Controlled Pre-Decision Tests

Constructing task-relevant prototypes first requires identifying which agent attributes produce meaningful decision differences in the current task. For each task, we provide the LLM with the task description, a dictionary of quantifiable external-context fields, and a dictionary of candidate agent attributes, and ask it to propose candidate factor–attribute pairs

$$\mathcal{G} = \{(z_j, a_j)\}_{j=1}^{J}, \tag{6}$$

where z_j is an actionable factor defined from the task, policy, event, or other external environment, and a_j is an agent attribute that may modulate the effect of that factor.

Candidate pairs are manually reviewed before testing. The review checks whether their semantics are clear, whether low and high levels can be defined numerically in a reproducible way, and whether multiple candidates are merely redundant descriptions of the same mechanism. Candidates with ambiguous value construction or highly redundant semantics are excluded from subsequent tests.

For each retained pair (z_j, a_j), we construct H_j base contexts. Within each base context, all non-target agent attributes, non-target external fields, decision-relevant dynamic states, and other textual facts are held fixed, while only the target attribute a_j and external factor z_j are varied. Let

$$a_j^{\ell}, \qquad z_j^{v}, \qquad \ell, v \in \{0, 1\}, \tag{7}$$

where 0 and 1 denote the reviewed low and high levels, respectively. Each base context therefore yields four controlled combinations.

Let $\boldsymbol{x}_{j,b}^{(\ell)}$ and $\boldsymbol{e}_{j,b}^{(v)}$ denote the complete inputs for base context b after setting the target attribute and external factor to the corresponding levels, and let $s_{j,b} \in \mathcal{S}$ denote the decision-relevant dynamic state held fixed across the four combinations. To reduce sensitivity to single-sample LLM variation, each cell is queried U times, where U is odd, and the modal output is used as the stable pre-decision:

$$\widehat{y}_{j,b,\ell,v} = \mathrm{Mode}_{r=1}^{U}\, \mathcal{O}_{\mathrm{LLM}}\Big(\boldsymbol{x}_{j,b}^{(\ell)}, \boldsymbol{e}_{j,b}^{(v)}, s_{j,b}\Big), \qquad U \text{ is odd}, \tag{8}$$

where $b = 1, \ldots, H_j$ and $\widehat{y}_{j,b,\ell,v} \in \mathcal{Y}$.

To capture both whether the decision category changes and the magnitude of that change, we define an ordinal mapping over decision states,

$$\nu : \mathcal{Y} \to \{0, \ldots, M-1\}, \qquad \nu(y^{(k)}) = k - 1. \tag{9}$$

For candidate pair j, the decision-state change rate is

$$\mathrm{CR}_j = \frac{1}{2H_j} \sum_{b=1}^{H_j} \sum_{v \in \{0,1\}} \mathbb{I}[\widehat{y}_{j,b,0,v} \neq \widehat{y}_{j,b,1,v}], \tag{10}$$

which measures how often changing the target agent attribute changes the discrete decision category while the external-factor level is fixed.

We further define the normalized mean change magnitude

$$\mathrm{MG}_j = \frac{1}{2H_j} \sum_{b=1}^{H_j} \sum_{v \in \{0,1\}} \frac{|\nu(\widehat{y}_{j,b,1,v}) - \nu(\widehat{y}_{j,b,0,v})|}{M-1}, \tag{11}$$

which measures the average ordinal movement caused by changing the attribute.

The combined task-relevance score of candidate pair j is

$$S_j = \alpha\, \mathrm{CR}_j + (1-\alpha)\, \mathrm{MG}_j, \qquad \alpha \in [0,1]. \tag{12}$$

Here, α controls the tradeoff between the frequency and magnitude of decision-state changes.

These tests are used only to identify agent attributes that are likely to affect discrete decisions in the current task; directional consistency, interaction direction, and statistical significance are not used as screening criteria. If the same agent attribute appears in multiple reviewed candidate pairs, we first average its corresponding CR values and MG values separately, then compute the final attribute score using Eq. (12).

Let $P_{\max}$ be the maximum allowed number of prototypes. We select the top

$$m = \lfloor \log_2 P_{\max} \rfloor \tag{13}$$

agent attributes, denoted $a_1, \ldots, a_m$. For selected attribute k, the median γ_k in the training population is used as the partition threshold:

$$b_{i,k} = \mathbb{I}[x_{i,a_k} > \gamma_k]. \tag{14}$$

The prototype code of agent i is

$$g(i) = (b_{i,1}, \ldots, b_{i,m}) \in \{0,1\}^m. \tag{15}$$

Thus, at most $2^m \leq P_{\max}$ task-relevant prototypes are formed. Each prototype corresponds to a combination of high/low intervals for the selected attributes rather than a predefined demographic label. Empty prototypes are excluded from program generation, while prototypes with insufficient samples are explicitly recorded and handled separately according to the experimental configuration.

B.2 Rule Sampling, Consolidation, and Use

For each nonempty prototype p, let $\mathcal{D}_p$ denote the set of reference LLM decision cases assigned to that prototype in which an actual decision update occurs. Each case contains the corresponding agent attributes, external context, decision-relevant dynamic state, and reference LLM decision, matching the inputs in Eq. (1). Natural-language rules provide behavioral priors for generating candidate program structures; they do not directly replace the program during online simulation.

To limit the number of cases used for rule induction, we define the rule-extraction budget for prototype p as

$$B_p = \lfloor \rho |\mathcal{D}_p| \rfloor, \qquad \rho \in (0, 1), \tag{16}$$

where ρ controls the fraction of reference cases used for induction.

Within this budget, cases are sampled for diversity rather than purely by occurrence frequency. Sampling prioritizes coverage of different external contexts, decision-relevant dynamic states, reference decisions, directions of decision change, and attribute ranges within the prototype, reducing the risk that the induced rules describe only high-frequency behaviors.

Each sampled batch is given to the LLM to induce behavior rules that are as atomic as possible. Each rule describes a limited set of applicability conditions and the corresponding decision response, avoiding overly long rules that conflate independent mechanisms.

Before entering the rule pool, new rules are semantically consolidated with existing ones. Synonymous or approximately equivalent rules merge their supporting examples; rules with narrower applicability but the same behavioral mechanism are recorded as specializations; rules with different condition structures or response mechanisms are retained as new rules.

Rule induction for a prototype stops when several consecutive batches produce no new behavioral mechanism or when the budget B_p is exhausted. The final rules are organized by support into three types: global rules that occur across prototypes, shared rules that recur in multiple prototypes, and prototype-specific rules that appear in only one prototype.

These rules are used only as structural guidance during candidate program generation, for example to help the LLM focus on potentially important variables, thresholds, conditional gates, and feature interactions. They are not used directly during online propagation and should not be interpreted as causal effects, statistical-significance results, or deterministic descriptions of individual behavior.

B.3 Input Normalization, Decision-State Quantization, and AST Constraints

Numerical static agent attributes and external-context features are normalized using ranges estimated from the training data. Decision-relevant dynamic states $s_{i,t}$ are provided directly according to their task-defined finite-state encoding and are not re-normalized as continuous values inside the program. For any numerical feature z requiring normalization, we define

$$\widetilde{z} = \begin{cases} \dfrac{z - z_{\min}^{\text{train}}}{z_{\max}^{\text{train}} - z_{\min}^{\text{train}}}, & z_{\max}^{\text{train}} > z_{\min}^{\text{train}}, \\ 0, & z_{\max}^{\text{train}} = z_{\min}^{\text{train}}. \end{cases} \tag{17}$$

where $z_{\min}^{\text{train}}$ and $z_{\max}^{\text{train}}$ are computed only from training data. Once fixed, these ranges remain unchanged during program optimization, region compilation, and online propagation; normalization is not reimplemented inside individual program expressions.

The decision-state quantizer $Q_{\boldsymbol{\tau}_p}$ uses $M-1$ ordered thresholds

$$\boldsymbol{\tau}_p = (\tau_{p,1}, \ldots, \tau_{p,M-1}) \tag{18}$$

to map a continuous program score to one of the M discrete decision states:

$$Q_{\boldsymbol{\tau}_p}(u) = y^{\left(1 + \sum_{k=1}^{M-1} \mathbb{I}[u > \tau_{p,k}]\right)}, \qquad \tau_{p,1} \le \cdots \le \tau_{p,M-1}. \tag{19}$$

Thus, the number of thresholds crossed by the continuous score u determines the ordinal decision state in $\mathcal{Y}$. Note that $Q_{\boldsymbol{\tau}_p}$ quantizes the program output y, not the decision-relevant dynamic state s, which is maintained by the task-specific state-update rule.

Candidate decision programs are represented as constrained abstract syntax trees (ASTs). Allowed primitive nodes include constants, input variables, continuous parameters, arithmetic operators, comparisons, logical operators, conditional branches, and a finite set of pre-approved numerically stable mathematical functions. This grammar supports linear and nonlinear responses, threshold gating, piecewise behavior, and limited feature interactions.

To ensure that generated programs are executable, deterministic, side-effect free, and auditable, the syntax validator prohibits loops, recursion, random-number generation, file-system access, network access, external imports, stateful global variables, unapproved functions, and any operation that may modify inputs or external state.

Let C_p be the number of continuous coefficients to optimize in the program for prototype p. Because M discrete decision states require $M-1$ quantization thresholds, if B_θ is the total parameter budget for continuous coefficients and decision-state thresholds, we require

$$C_p + M - 1 \leq B_\theta. \tag{20}$$

Candidate programs first pass static syntax checks before parameter optimization. Programs exceeding the total parameter budget, maximum AST node count, or maximum tree depth are discarded. These constraints are used only to bound the search space and ensure safety and auditability; they do not change the behavioral fitting objective defined in the main text.

B.4 Parameter Optimization and Structural Search

Table 4: Experimental configuration for program generation, island search, and numerical parameter optimization.

Configuration	Value
Number of search islands	4
Candidates generated per LLM request	2
Maximum evolutionary generations	6
Maximum distinct candidates retained per prototype	48
Elite migration interval and count	1 candidate every 2 generations
Stagnation patience	3 generations; at most 1 restart per island
Total budget for coefficients and thresholds	$C_p + M - 1 \leq 32$
Maximum AST nodes and depth	120; 12
Coefficient and threshold bounds	$c \in [-8, 8]$; $\tau \in [-5, 5]$
One-step / closed-loop Differential Evolution max iterations	3; 3
Initial search radius (one-step / closed-loop)	1.0; 0.45
Mutation range, recombination probability, polish	$[0.5, 1.0]$; 0.7; disabled
Weight of one-step error in closed-loop objective	0.15
Max parent programs / rule-prompt limit	3; 24 rules
Candidate programs per prototype / coordinate-optimization rounds	5; at most 4 rounds

For a fixed candidate program structure of prototype p, let

$$\boldsymbol{\theta}_p = (\boldsymbol{c}_p, \boldsymbol{\tau}_p) \tag{21}$$

denote all parameters to be calibrated, where $\boldsymbol{c}_p$ contains the continuous coefficients inside the program and $\boldsymbol{\tau}_p$ contains the decision-state quantization thresholds.

Parameter optimization first constrains one-step fit to reference LLM decisions. For the reference decision cases $\mathcal{D}_p$ of prototype p, define the one-step ordinal decision-state error as

$$\mathcal{L}_{\text{one}}(\boldsymbol{\theta}_p) = \frac{1}{|\mathcal{D}_p|} \sum_{d \in \mathcal{D}_p} \frac{|\nu(\widehat{y}(d; \boldsymbol{\theta}_p)) - \nu(y^+(d))|}{M-1}, \tag{22}$$

where $y^+(d)$ is the post-update decision state produced by the reference LLM for case d, and $\widehat{y}(d; \boldsymbol{\theta}_p)$ is the candidate program output under the same agent attributes, external context, and decision-relevant dynamic-state input.

The first optimization stage minimizes Eq. (22) to obtain a reasonable local decision response. Because low one-step error does not guarantee alignment after repeated execution, the second stage performs closed-loop calibration.

During closed-loop replay, the candidate program takes $s_{i,t-1}$ as the dynamic-state input at timestep t and produces $\widehat{y}_{i,t}$. The decision-relevant dynamic state $s_{i,t}$ is then updated using the same task-specific rule $\mathcal{U}$ as in the reference simulation and becomes the input to the next program decision. Thus, multi-step calibration follows the dynamic-state trajectory induced by the candidate program's own previous decisions rather than repeatedly resetting to the reference LLM trajectory.

Let $\mathcal{T}_{\text{cal}}$ denote the closed-loop calibration window. We define

$$\mathcal{L}_{\text{roll}}(\boldsymbol{\theta}_p) = \frac{1}{|\mathcal{T}_{\text{cal}}|} \sum_{t \in \mathcal{T}_{\text{cal}}} \text{JSD}\big(\widehat{\pi}_{p,t}, \pi_{p,t}^{\text{LLM}}\big) + \lambda\, \mathcal{L}_{\text{one}}(\boldsymbol{\theta}_p), \qquad \lambda \geq 0, \tag{23}$$

where $\widehat{\pi}_{p,t}$ and $\pi_{p,t}^{\text{LLM}}$ are the decision-state distributions within prototype p produced by the candidate program and reference LLM, respectively, and JSD denotes Jensen–Shannon divergence. The first term constrains population-level behavior after multi-step execution, while the second preserves one-step fidelity so that the program cannot match aggregate proportions through unreasonable local decisions.

The second stage starts from the parameters found by one-step optimization and minimizes Eq. (23). Because the programs in Eq. (3) may contain conditional branches and the quantizer in Eq. (19) is discrete, the parameter objective is generally non-differentiable and discontinuous. We therefore use bounded gradient-free global optimization, specifically Differential Evolution, to calibrate continuous coefficients and decision-state thresholds. Thresholds are sorted or reparameterized to enforce

$$\tau_{p,1} \leq \tau_{p,2} \leq \cdots \leq \tau_{p,M-1}. \tag{24}$$

Structural search is performed in addition to continuous-parameter optimization. We use multiple independent search islands. Each island maintains a population of candidate programs and retains non-dominated candidates according to closed-loop error and expression complexity. At each generation, the LLM may use well-performing parent programs and the corresponding natural-language rules as context to propose new structures; each new program undergoes syntax validation before parameter optimization.

Search islands periodically exchange a small number of elite programs to share useful structures while maintaining diversity. If an island shows no improvement for several generations, stagnation is detected and a limited restart may be triggered. The programs produced by all islands form the final prototype-level candidate pool.

The main structural-search and numerical-optimization settings used in our experiments are summarized in Table 4. Implementation options not listed separately are held fixed across experiments and are not used as ablation variables.

After obtaining multiple candidate programs for each prototype, we do not choose the final program solely by one-step training error. Instead, we compare combinations of prototype programs on an independent selection set using closed-loop population behavior. Let $\mathcal{V}$ denote the model-selection window, and let Ω be one combination chosen from the prototype candidate pools. The selection objective is

$$\mathcal{L}_{\text{sel}}(\Omega) = \frac{1}{|\mathcal{V}|} \sum_{t \in \mathcal{V}} \text{JSD}\big(\widehat{\pi}_t^{\Omega}, \pi_t^{\text{LLM}}\big) + \beta\, \text{Complexity}(\Omega), \qquad \beta \geq 0. \tag{25}$$

Here, $\widehat{\pi}_t^{\Omega}$ is the aggregate decision-state distribution obtained by closed-loop execution of program combination Ω, π_t^{LLM} is the reference LLM distribution at the same timestep, and $\mathrm{Complexity}(\Omega)$ measures structural complexity, for example through program length, number of branches, and number of parameters. The coefficient β controls the tradeoff between behavioral fit and program simplicity.

We select the combination with low closed-loop error on the independent selection set subject to the complexity constraints. All selected task-relevant attributes, prototype thresholds, input-normalization ranges, program structures, continuous parameters, and decision-state quantization thresholds are then fully frozen before region compilation. Final test data are not used for candidate-factor screening, rule induction, program generation, parameter optimization, or program selection; they are used only for final evaluation after the model is frozen.

C Implementation Details of Region Compilation and Event-Driven Propagation

This appendix provides implementation details for region compilation and event-driven propagation, including adaptive region construction, region-level transition matrices, population-count propagation under different activation semantics, temporal-block boundaries, block-level updates, and feedback handling. The main method uses only the compiled regions and propagation operators produced by these procedures.

C.1 Adaptive Region Construction

After behavior compilation, all prototype programs and their parameters are frozen. For prototype p, region construction considers only the static agent attributes actually accessed by F_p, together with any raw attributes required by derived features used in the program.

Let $\mathcal{E}_p \subseteq \mathcal{E}_{\mathrm{ctx}}$ denote the set of complete external contexts that prototype p may encounter during execution. For any candidate region R, let $\mathcal{S}_R^{\mathrm{cal}}$ be the calibration agents that fall into that region. For each context $\boldsymbol{e} \in \mathcal{E}_p$, we define the region-level state-transition matrix $P_R^{(\boldsymbol{e})} \in [0,1]^{K\times K}$ by

$$P_R^{(\boldsymbol{e})}[r,r'] = \frac{1}{\left|\mathcal{S}_R^{\mathrm{cal}}\right|} \sum_{i\in\mathcal{S}_R^{\mathrm{cal}}} \mathbb{I}\Big[U\Big(s^{(r)}, F_p(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{e}}, s^{(r)}; \theta_p), \boldsymbol{e}\Big) = s^{(r')}\Big], \quad r,r' \in \{1,\dots,K\}. \tag{26}$$

To measure how heterogeneous the program responses are within a region, we define the mean impurity

$$\mathrm{Imp}(R) = \frac{1}{K|\mathcal{E}_p|} \sum_{\boldsymbol{e}\in\mathcal{E}_p} \sum_{r=1}^{K} \Big(1 - \max_{r'} P_R^{(\boldsymbol{e})}[r,r']\Big). \tag{27}$$

When most agents in a region transition to the same successor state under a given pre-update dynamic state and external context, $\mathrm{Imp}(R)$ is small. It increases as the program-induced transitions become more dispersed.

For a candidate binary split $R \to (R_{\mathrm{L}}, R_{\mathrm{R}})$, we define the impurity gain as

$$G(R) = \mathrm{Imp}(R) - \frac{|R_{\mathrm{L}}|\,\mathrm{Imp}(R_{\mathrm{L}}) + |R_{\mathrm{R}}|\,\mathrm{Imp}(R_{\mathrm{R}})}{|R|}. \tag{28}$$

Among the allowed attributes and split points, the algorithm chooses the valid split with the largest gain and recursively refines regions whose program responses remain heterogeneous. Splitting stops when the maximum number of leaves or maximum depth is reached, a child region contains too few samples, the impurity of the current region falls below a tolerance, or the best split gain is below a threshold.

Adaptive regions are therefore typically finer near program decision boundaries and coarser in attribute ranges where program responses are stable. The procedure queries only the frozen programs offline; it does not train a new classifier.

C.2 Region Transitions and Population-Count Propagation

For each final leaf region $\mathcal{R}_{p,\ell}$ of prototype p, let $\mathcal{S}^{\mathrm{cal}}_{p,\ell}$ denote its calibration agents. For every complete external context $\boldsymbol{e} \in \mathcal{E}_p$, we precompute $P^{(\boldsymbol{e})}_{p,\ell} \equiv P^{(\boldsymbol{e})}_{\mathcal{R}_{p,\ell}}$ according to Eq. (4). These transition matrices remain frozen throughout online propagation.

Every external context reachable during execution must be included in $\mathcal{E}_p$ in complete-field form before region compilation. If endogenous feedback can generate additional contexts, these must also be compiled. If an uncovered complete context appears at runtime, the system constructs the corresponding transition matrix using the already frozen behavioral programs, or falls back to per-agent execution of those frozen programs. It does not invoke the reference LLM again or infer new context semantics by substituting undefined partial fields.

During online execution, each prototype–region unit is represented by the dynamic-state population vector $\boldsymbol{m}_{p,\ell,t} \in \mathbb{R}^K_{\geq 0}$ defined in the main text, whose r-th component is the number of agents in state $s^{(r)}$. We first consider without-replacement activation, under which each agent can be activated at most once within an activation round.

For unit (p, ℓ), let $\boldsymbol{w}_{p,\ell,t}$ and $\boldsymbol{c}_{p,\ell,t}$ denote the dynamic-state counts that have not yet been updated and those that have already been updated in the current round, respectively, with

$$\boldsymbol{m}_{p,\ell,t} = \boldsymbol{w}_{p,\ell,t} + \boldsymbol{c}_{p,\ell,t}. \tag{29}$$

If $n_{p,\ell,t}$ agents must be activated at timestep t, they are drawn from the not-yet-updated mass in proportion to the current dynamic-state distribution:

$$\boldsymbol{u}_{p,\ell,t} = \begin{cases} \dfrac{\bar{n}_{p,\ell,t}}{\boldsymbol{w}_{p,\ell,t}\mathbf{1}_K}\boldsymbol{w}_{p,\ell,t}, & \boldsymbol{w}_{p,\ell,t}\mathbf{1}_K > 0, \\ \mathbf{0}, & \boldsymbol{w}_{p,\ell,t}\mathbf{1}_K = 0, \end{cases} \tag{30}$$

where

$$\bar{n}_{p,\ell,t} = \min(n_{p,\ell,t}, \boldsymbol{w}_{p,\ell,t}\mathbf{1}_K)\,. \tag{31}$$

The state counts are then updated under the current complete external context $\boldsymbol{e}_t$ as

$$\boldsymbol{w}_{p,\ell,t+1} = \boldsymbol{w}_{p,\ell,t} - \boldsymbol{u}_{p,\ell,t}, \tag{32}$$

$$\boldsymbol{c}_{p,\ell,t+1} = \boldsymbol{c}_{p,\ell,t} + \boldsymbol{u}_{p,\ell,t}P^{(\boldsymbol{e}_t)}_{p,\ell}. \tag{33}$$

At the end of a without-replacement activation round, the updated mass is returned to the waiting pool:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{c}, \qquad \boldsymbol{c} \leftarrow \mathbf{0}. \tag{34}$$

The approximation in this propagation comes from within-region aggregation: agents with different exact attributes but assigned to the same region share a common conditional transition distribution. The procedure therefore approximates the dynamic-state propagation induced by per-agent programs without attempting to reconstruct each individual agent's future trajectory.

C.3 Temporal-Block Boundaries and Program-Relevant Context

Event-driven propagation splits the timeline only when conditions that affect region updates change. Because a prototype program typically uses only a subset of the complete external context, a change in an irrelevant field does not require a new temporal block.

Let $\mathcal{F}^{\mathrm{ext}}_p$ denote the external-context fields actually accessed by prototype program F_p, including raw fields required by any derived variables. We define the program-relevant context signature at timestep t as

$$\kappa_p(t) = \mathrm{Serialize}\big(\{e_{t,j} \mid j \in \mathcal{F}^{\mathrm{ext}}_p\}\big)\,. \tag{35}$$

If $\kappa_p(t)$ remains unchanged, external context does not change the region transition dynamics for prototype p. Temporal blocks are also forced to end when the activation probability or activation

round changes, at requested query times, at predefined feedback-review times, and at the end of the simulation.

A temporal block therefore represents a contiguous interval over which both the propagation operator and the activation semantics remain stable.

C.4 Block-Level Propagation with Without-Replacement Activation

Consider a temporal block $[t_0, t_1]$ with fixed propagation conditions, entirely contained within a single without-replacement activation round. Define the cumulative number of activations for a region over the block as

$$n_{p,\ell,[t_0,t_1]} = \sum_{t=t_0}^{t_1} n_{p,\ell,t}. \tag{36}$$

If the same region transition matrix $P_{p,\ell}$ applies throughout the block, the total mass to update can be written directly as

$$\boldsymbol{u}_{p,\ell,[t_0,t_1]} = \begin{cases} \dfrac{\bar{n}_{p,\ell,[t_0,t_1]}}{\boldsymbol{w}_{p,\ell,t_0}\mathbf{1}_K}\boldsymbol{w}_{p,\ell,t_0}, & \boldsymbol{w}_{p,\ell,t_0}\mathbf{1}_K > 0, \\ \mathbf{0}, & \boldsymbol{w}_{p,\ell,t_0}\mathbf{1}_K = 0, \end{cases} \tag{37}$$

where

$$\bar{n}_{p,\ell,[t_0,t_1]} = \min\left(n_{p,\ell,[t_0,t_1]}, \boldsymbol{w}_{p,\ell,t_0}\mathbf{1}_K\right). \tag{38}$$

The state at the end of the block is

$$\boldsymbol{w}_{p,\ell,t_1+1} = \boldsymbol{w}_{p,\ell,t_0} - \boldsymbol{u}_{p,\ell,[t_0,t_1]}, \tag{39}$$

$$\boldsymbol{c}_{p,\ell,t_1+1} = \boldsymbol{c}_{p,\ell,t_0} + \boldsymbol{u}_{p,\ell,[t_0,t_1]}P_{p,\ell}. \tag{40}$$

Under proportional sampling by dynamic state, a constant transition matrix within the block, at most one activation per agent in the current round, and no autonomous state changes for inactive agents, Eq. (40) yields the same region-level state counts at the block endpoint as sequentially applying Eqs. (30)–(33).

C.5 Repeated Activation and Feedback Propagation

For execution modes in which the same agent may be reactivated at different timesteps, the framework supports independent homogeneous Bernoulli activation within a region. If the activation probability per step is q over a stable block and the region transition matrix is P, the expected one-step propagation operator is

$$Q = (1-q)I_K + qP. \tag{41}$$

If the same propagation conditions hold for k consecutive timesteps, we can compute

$$\boldsymbol{m}_{p,\ell,t+k} = \boldsymbol{m}_{p,\ell,t}Q^k, \tag{42}$$

thereby avoiding identical matrix updates at every timestep. This equivalence assumes homogeneous, independent activation within the region; an arbitrary pre-specified repeated-activation sequence for particular agents is not covered by this mode.

The framework also allows endogenous external feedback to be triggered from aggregate states at predefined review times. Let

$$\boldsymbol{G}_{t_{\mathrm{rev}}} = \sum_{p,\ell} \boldsymbol{m}_{p,\ell,t_{\mathrm{rev}}} \tag{43}$$

denote the aggregate dynamic-state count vector at review time t_{rev}.

Feedback rules are usually defined over the decision states of interest rather than the full extended dynamic state. For a set of decision states $A \subseteq \mathcal{Y}$, let $\mathcal{S}(A) \subseteq \mathcal{S}$ denote the set of dynamic states whose decision component belongs to A. A feedback trigger can then be written as

$$\frac{\sum_{r:\, s^{(r)} \in \mathcal{S}(A)} G^{(r)}_{t_{\mathrm{rev}}}}{\boldsymbol{G}_{t_{\mathrm{rev}}}\mathbf{1}_K} \geq \eta, \tag{44}$$

where $\eta \in [0, 1]$ is a predefined threshold.

The feedback review is performed after state propagation at the current timestep, and any newly triggered external context takes effect from the next timestep. Every review time therefore forms a temporal-block boundary. All complete external contexts that can arise from feedback must also be included in the context set during region compilation.

Under these conditions, event-driven block computation is numerically equivalent at block endpoints to stepwise execution of the same region-level propagation. The remaining end-to-end error therefore comes primarily from the preceding behavioral-program approximation and region aggregation rather than from temporal-block computation itself.

C.6 Computational Complexity and Error Analysis

Error analysis. Behavior programs approximate reference decisions, while regional averaging can discard attribute-state dependence by replacing state-conditional transitions with calibration averages. Let $\epsilon_{\text{program}}$ and ϵ_{region} uniformly bound, in total variation (TV), the effective one-step discrepancies between reference and individual-program transitions and between state-conditional program transitions and regional operators, respectively, including activation. For identical initial joint attribute-state distributions and exogenous context sequences, with synchronous or independent homogeneous activation within regions, the regional output and expected reference output satisfy

$$\mathrm{TV}(\widehat{\pi}_t, \mathbb{E}[\pi_t]) \le \min\{1,\ t(\epsilon_{\text{program}} + \epsilon_{\text{region}})\}. \tag{45}$$

This conservative bound follows from a stepwise error decomposition and the nonexpansiveness of Markov propagation in TV. The one-step bounds cannot be replaced directly by average training error or regional impurity, and the result does not automatically cover context sequences that diverge through feedback. Aggregation is exact when calibration and deployment agents within each region have identical transitions across all reachable states and contexts, but partitioning under a finite budget does not guarantee this condition. With stable propagation and activation rules and preserved event, reset, and feedback boundaries, temporal merging remains equivalent to stepwise propagation of the same region model, up to floating-point roundoff.

Synthetic diagnostic. Figure 3 isolates aggregation error without program approximation or calibration sampling error. With heterogeneous-type proportion $h = 0.01$, mean impurity is only 0.005, yet JSD reaches 0.7619 after 256 steps. Holding both type and initial-state marginals fixed, increasing the initial correlation ρ from 0 to 1 increases final-step JSD from 0.1379 to 0.3113. Refinement by known behavioral type restores exact population propagation in every setting. A separate control with transitions independent of the previous state also remains exact despite perfect attribute-state correlation. Thus, low average impurity does not ensure long-horizon fidelity; the relevant mechanism is a mismatch in effective conditional transitions. This stress test characterizes failure under deliberately coarse aggregation, rather than estimating error on the empirical datasets.

Computational complexity. Let A denote the total number of activated agent decisions, T the number of simulation steps, R the number of regions, K the number of dynamic states, and U the total number of time-block updates across all regions. Behavior compilation reduces the decision cost from $O(Ac_{\text{LLM}})$ to $O(Ac_F)$, where c_{LLM} and c_F are the costs of one LLM decision and one program evaluation, respectively. Region sharing replaces individual execution with dense state propagation at a cost of $O(RTK^2)$. Once block operators have been prepared, temporal merging further reduces the propagation cost to $O(UK^2)$. Finer regions help preserve individual differences but increase R, while richer memory encodings help retain decision-relevant information but may increase K. The benefit of temporal compression depends on the lengths of stable intervals and the frequency of output queries: when outputs are required at every step, $U = RT$, eliminating this benefit. Full computational costs must also account for one-time compilation, population mapping, region-operator construction, query aggregation, and matrix-power preparation. In particular, preparing a repeated-activation block of length $\Delta \ge 2$ typically requires $O(K^3 \log \Delta)$ work using dense matrix exponentiation. The computational advantage therefore depends on the reusability of behavior programs, the reduction in the number of regions relative to the population size, and the temporal stability of propagation conditions.

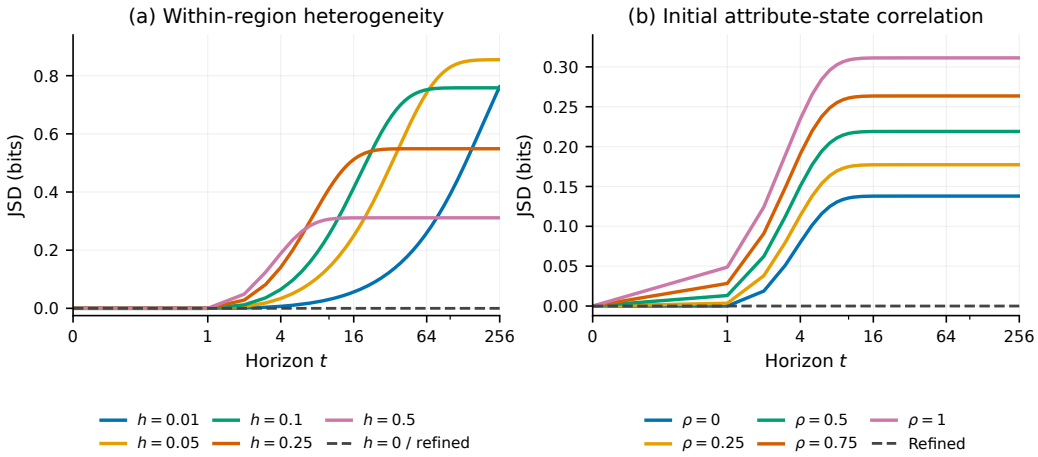


Figure 3: Synthetic aggregation stress test with $N = 10{,}000$, $T = 256$, two states, and synchronous updates. Type A preserves its state; type B, with population fraction h, always enters state 1. (a) All agents start in state 0. (b) With $h = 0.5$, set $\Pr(s_0 = 1 \mid B) = (1 + \rho)/2$ and $\Pr(s_0 = 1 \mid A) = (1 - \rho)/2$, preserving an initial state-1 fraction of 0.5 and making ρ the initial Pearson correlation. The complete population supplies calibration transitions. All timesteps are plotted on a symmetric-logarithmic horizontal axis; JSD uses base-2 logarithms. Refinement uses known A/B types and is not a guarantee about the heuristic partitioner.

D Supplementary Experimental Settings and Analysis

D.1 Control Settings and Statistical Interpretation

The ablation experiments use the same data splits and experimental configuration as the main experiments, with three independent compilations per setting. The SoP rows reuse the same three complete compilations as the main experiments rather than generating a separate set of programs specifically for the ablations. For each ablation, only the target component is replaced or removed; upstream outputs and settings that are not directly intervened on remain unchanged. This design isolates the effect of the component under study as much as possible.

The behavior-compilation ablations include three variants. A1 replaces the task-relevant attribute partition with preset attributes to assess the role of data-driven partitioning in modeling behavioral heterogeneity. A2 removes rule guidance during program construction to measure the contribution of structural priors to program search. A3 retains program generation but removes closed-loop parameter fitting, isolating the benefit of the multi-step rollout objective relative to local one-step fitting. All three variants are evaluated through closed-loop per-agent program replay using JSD and Exact Match.

Propagation ablations are performed after the behavioral model has been fixed. A4 disables state-propagation sharing across agents within a region: each agent maintains its own probability state and advances independently using the frozen transition operator of its assigned region. A5 retains region sharing but disables merging across consecutive timesteps, applying the transition operator at every step. These controls measure the computational benefit of spatial sharing and temporal merging, respectively. A4 retains SoP's temporal merging and block boundaries. Population, events, initial conditions, transition operators, and activation semantics are fixed across variants, and each query schedule uses identical output points for all three variants. For Table 2, Reddit propagation JSD is evaluated in runs with every-step outputs, while Updates are measured in separate runs with queries at the end of each eight-step evidence block over the full 1,024-step horizon. WVS uses queries after each of its eight rounds for both measures. These controls isolate differences in execution rather than approximation error from region construction or transition learning.

A3 changes only the parameter-fitting objective and does not remove program-structure generation. On WVS, we keep the program family, candidate count, and maximum program-evaluation budget unchanged, replacing the second-stage closed-loop objective with one-step error; if the optimizer

converges early, no artificial evaluations are added. Reddit similarly preserves the program structure, one-step initialization, training/calibration split, and second-stage program-evaluation budget of the full method, but replaces the closed-loop calibration objective with calibration-set one-step error plus 0.15 times the training-set one-step error, and selects numerical restarts under this objective. This setup minimizes confounding from changes in data scope, program structure, or extra parameter evaluations. The JSD and Exact Match values in Table 2 report the mean and sample standard deviation over the three compilations.

D.2 Additional Behavior-Compilation Results

On WVS, SOP obtains a JSD of 0.1289. Replacing task-relevant partitioning with preset attributes raises A1 JSD to 0.1694, an increase of 0.0405, while Exact Match falls from 67.02% to 65.19%, a decrease of 1.83 percentage points. Removing rule guidance yields an A2 JSD of 0.1514, 0.0225 above SOP, and lowers Exact Match to 65.05%, a decrease of 1.97 percentage points. A3 shows the largest degradation: JSD rises to 0.1796, 0.0507 above SOP, while Exact Match drops by 15.50 percentage points to 51.52%.

Reddit shows the same overall pattern. A1 increases JSD from 0.2658 to 0.2875 (+0.0217) and lowers Exact Match from 48.08% to 45.72%. A2 produces a smaller JSD increase, from 0.2658 to 0.2855 (+0.0197), and an Exact Match of 47.99%, only 0.09 percentage points below SOP. In contrast, A3 reaches a JSD of 0.3076, an increase of 0.0418 over SOP, while Exact Match falls by 5.19 percentage points to 42.89%.

Together, the two datasets show that the three mechanisms contribute differently to final program-replay quality. Removing closed-loop fitting causes the largest degradation in both JSD and Exact Match, indicating that feedback from multi-step execution is critical for correcting long-run program behavior. Task-relevant attribute partitioning yields a consistent but more modest improvement, suggesting that reducing within-program behavioral heterogeneity helps produce more coherent dynamics. Rule guidance has a smaller and more dataset-dependent effect, but removing it raises JSD on both datasets, indicating that structured search priors still improve the final programs.

D.3 Updates and Query Granularity

We count advancing the probability-state vector of one propagation entity by one temporal block or one timestep as one *Update*, including the required initial state initialization. If a batched operation advances k propagation entities simultaneously, it counts as k Updates. Let $\mathcal{G}$ be the set of propagation entities and let B_g denote the number of advances executed by entity g over the full simulation. The total number of Updates is

$$C_{\mathrm{upd}} = \sum_{g \in \mathcal{G}} B_g. \tag{46}$$

In SOP, propagation entities are regions. In A4, they are individual agents. In A5, they remain regions but are advanced one timestep at a time. Updates are recorded directly from the actual propagation loop and are not further weighted by population size or state dimension. The metric excludes LLM compilation, transition-operator construction, matrix-power preparation, population mapping, query aggregation, and file output. It therefore measures the core amount of state advancement performed during propagation rather than end-to-end program cost.

The three complete WVS compilations produce 2, 31, and 22 nonempty deployment regions. With queries after all eight rounds, the corresponding numbers of Updates are 16, 248, and 176, averaging 146.7 and rounded to 147 in the main table. A4 advances each of $1{,}000$ agents eight times and therefore requires $8{,}000$ Updates for each compilation. Relative to per-agent propagation, SOP reduces mean Updates by approximately 98.2%. A5 retains region sharing, but WVS contains an event change and a query boundary in every round, leaving no additional opportunity for temporal merging; its Update count is therefore identical to SOP. All three execution paths have JSD 0.1295 ± 0.0589 at the same query points, and the maximum probability difference between A4 and SOP is below 1.13×10^{-14}, confirming numerical equivalence under fixed transition operators.

Reddit contains 100 agents, 8 regions, and 128 eight-step evidence blocks. When querying at the end of each evidence block, SOP executes $8 \times 128 = 1{,}024$ region Updates; A4 advances 100

Table 5: Propagation cost under different query granularities on Reddit. JSD is computed at the query points in each row; the three execution paths are numerically equivalent at the same query points.

Query Setting	Query Points	JSD	SOP	A4	A5
Every step	1,024	0.2686 ± 0.0143	8,192	102,400	8,192
Every 8-step block	128	0.3033 ± 0.0145	1,024	12,800	8,192
Final state only	1	0.0750 ± 0.0018	1,024	12,800	8,192

Note: The SOP, A4, and A5 columns all report Updates.

individual agents across 128 blocks for $100 \times 128 = 12{,}800$ Updates; and A5 advances 8 regions step by step over all $1{,}024$ timesteps for $8 \times 1{,}024 = 8{,}192$ Updates. Thus, spatial sharing reduces Updates to $1/12.5$ of per-agent execution, and temporal merging further reduces Updates to $1/8$ of stepwise region propagation. Full results under different query granularities are shown in Table 5.

When every timestep must be queried, both SOP and A5 require $8{,}192$ Updates because any cross-step merging would skip required intermediate outputs; temporal merging is therefore impossible. When queries occur only at the end of each eight-step evidence block, SOP needs only $1{,}024$ Updates while A5 still performs $8{,}192$ stepwise Updates. With only the final state queried, SOP still requires $1{,}024$ Updates because topic resets and evidence updates remain mandatory boundaries that prevent the full $1{,}024$-step horizon from collapsing into a single propagation.

On Reddit, Table 2 reports propagation JSD from every-step evaluation (0.2686 ± 0.0143), while its Updates are counted in separate runs that request only the 128 block-end outputs. Table 5 reports both JSD and Updates under the query schedule specified in each row; its block-end JSD is 0.3033 ± 0.0145. These JSD values average over different sets of timesteps and do not indicate a difference in propagation fidelity at shared query points. The Updates in Table 2 therefore do not measure the cost of generating the full trajectory used for its JSD evaluation. Both parts of Table 2 evaluate JSD over the full horizon: Part A uses per-agent program replay, whereas Part B uses expected population propagation.

Under the eight-step block-end query setting, ten post-warmup timing runs yield an A5-to-SOP runtime ratio of about $3.50\times$, below the $8\times$ ratio in Updates. Wall-clock time includes loop control, query aggregation, and other fixed overheads that do not scale proportionally with Updates. This further illustrates that Updates is a mechanism-level measure of propagation work rather than a direct end-to-end speedup factor.

D.4 Additional Control for Within-Prototype Region Merging

A4 keeps each region's transition operator fixed, so it measures only the computational benefit of propagation sharing and cannot directly determine whether region aggregation itself loses fidelity. To separate these questions, we introduce an additional "one region per prototype" control that merges all regions within each prototype and evaluates the resulting change in the population distribution.

On WVS, merging region transition operators within each prototype according to their frequency in the training population increases deployment JSD from 0.1295 to 0.1427, a rise of 0.0133. This result indicates that, under the current setting, retaining finer region partitions within a prototype helps limit approximation error from aggregation. The propagation-sharing gain measured by A4 and the fidelity effect of region granularity should therefore be treated as separate questions: the former comes from sharing one operator across multiple agents, whereas the latter depends on how much heterogeneous behavior is aggregated into one propagation region.

All three complete Reddit compilations produce eight regions in one-to-one correspondence with the eight nonempty prototypes, so "one region per prototype" does not change the partition. Under every-step queries, the mean deployment JSD is 0.2686 both before and after merging. The current Reddit setting therefore cannot separately test whether further within-prototype region refinement would improve fidelity. The $12.5\times$ Update compression observed in A4 only demonstrates the sharing benefit of eight regions relative to 100 individual agents; it does not establish the effect of finer region partitioning on behavioral accuracy.

E SUPPLEMENTARY SETTINGS AND RESULTS FOR COMPILER-LLM ROBUSTNESS

E.1 DATA SCOPE AND INTERVENTION SETUP

This experiment fixes the existing Full–LLM reference trajectories and changes only the model used for every actual LLM call in behavior compilation, including the controlled pre-decisions used for prototype extraction. Region construction, transition operators, and subsequent propagation are recomputed from the newly generated programs; no learned artifact from another compiler model is reused. We evaluate nine models, each with three independent compilations on both datasets, for a total of $9 \times 2 \times 3 = 54$ planned experimental units. All runs complete successfully, and the summary does not select the best compilation repeat.

WVS uses two training runs, one validation run, and one test run. We fix a sample of 100 agents from each run and retain all eight rounds. Sampling is fixed before the experiment and depends only on predetermined run and agent indices, not on behavioral labels or method performance. The two training runs contain $1{,}600$ decisions in total, while the validation and test sets contain 800 decisions each. Because the original runs may draw from overlapping profile pools, the split does not guarantee complete identity disjointness.

Reddit uses two training runs, one calibration run, and one test run. Each run retains the same 100 agents and the first two complete topics. Each topic spans 32 timesteps with evidence updated every 8 steps, giving an evaluation horizon of 64 steps and 8 evidence blocks while preserving the original activation schedule, topic resets, and agent-specific evidence memory. The two training runs contain 804 and 771 activated decisions, respectively; the calibration and test sets contain 818 and 774. Calibration topics are used for second-stage parameter fitting and numerical-restart selection and are not treated as independent test data. Test labels are used only after the behavioral programs are frozen. Region compilation may use known attributes of the target population and public external context, but not the test reference decisions.

E.2 BUDGETS AND MODEL INTERFACES

Within each dataset, all models share the same feature catalog, task prompt templates, case-sampling ratio, and optimization-budget rules. The maximum number of prototypes is 8, the rule-case sampling ratio is capped at 20%, and the program-parameter budget is 32.

For WVS, two candidate programs are generated per prototype. The Differential Evolution parameters $(\mathrm{maxiter}, \mathrm{popsize})$ are $(5, 4)$ for one-step fitting and $(3, 3)$ for closed-loop fitting, and the final program is selected by validation-set closed-loop JSD together with program complexity. For Reddit, one program structure is generated per prototype followed by four numerical restarts; the maximum iterations of the two fitting stages are 6 and 4, with population multiplier 3. The second-stage objective is calibration-set closed-loop JSD plus 0.15 times the training-set one-step error. These four restarts are numerical restarts of the same structural candidate, not four independent structural searches. Because compiler models may induce different prototype partitions, the actual number of examples, LLM Calls, tokens, and monetary cost can differ even under identical budget rules.

Some feature dimensions are constant over the two Reddit training-topic prefixes and therefore cannot form meaningful high/low contrasts in the prototype pre-decision test. These dimensions are excluded only from the prototype pre-decision candidate catalog; the full feature catalog remains available for rule induction and program generation. Region propagation uses a fixed Bernoulli activation probability $q = 0.125$ over 21 joint states that include evidence memory. JSD is computed after projecting these states to five attitude categories, with the state lacking a formed attitude mapped to neutral.

The actual API identifiers and reasoning settings are listed in Table 6. Because interfaces differ, GPT-6 Astra, Gemini 3.8 Flash, and GLM 5.3 Flash use low-intensity reasoning, while explicit reasoning is disabled for the other models. We adapt only parameters and structured-output requirements necessary for each interface; all returned outputs pass the same parsing and semantic validation, and failed requests are not substituted with another model. The experiment therefore evaluates

Table 6: Compiler-model identifiers and inference configurations.

Display Name	API Model Identifier	Reasoning Setting
DeepSeek V4.1 Flash	deepseek/deepseek-v4.1-flash	none
GPT-6 Astra	openai/gpt-6-astra	low
Gemini 3.8 Flash	google/gemini-3.8-flash	low
HY4 Preview	tencent/hy4-preview	none
GLM 5.3 Flash	z-ai/glm-5.3-flash	low
Qwen3.8 Flash	qwen/qwen3.8-flash	none
Kimi K3	moonshotai/kimi-k3	none
MiniMax M3	minimax/minimax-m3	none
MiMo-V2.5	xiaomi/mimo-v2.5	none

pipeline robustness under each model's available interface configuration rather than a model ranking under exactly matched reasoning tokens or monetary cost.

E.3 Metrics and Uncertainty

We evaluate per-agent program replay and region-level expected propagation separately. JSD in both cases is the base-2, non-square-rooted Jensen–Shannon divergence averaged over all evaluation timesteps. On WVS, Exact Match covers all 800 agent–round pairs in the test run. On Reddit, it covers only the 774 actually activated test decisions. Region-level expected propagation produces probability mass rather than a unique per-agent prediction label, so Exact Match is not computed for propagation.

The standard deviations in Table 3 are sample standard deviations across the three independent compilations. We additionally compute 95% intervals using $5{,}000$ paired bootstrap resamples of complete agent trajectories. Each resample shares the same agent weights across compiler models and compilation repeats; metrics are first computed for each compilation and then averaged across the three repeats. Region-propagation results are recomputed by reweighting the initial population's probabilistic contribution within each region. These intervals are conditional on the fixed behavioral programs and reference trajectories and therefore do not include uncertainty from recompilation. The Reddit intervals are also conditional on the two fixed test topics and do not estimate between-topic variation. We therefore do not interpret point-estimate rankings of individual compiler models as universal or statistically significant superiority.

To compare program replay with region propagation, we first average program JSD and deployment JSD over the three compilations for each model, take the absolute difference, and then average those differences equally over the nine models. The mean absolute differences are 0.0025 on WVS and 0.0050 on Reddit, with maximum model-level absolute differences of 0.0061 and 0.0124, respectively. This quantity measures how the error relative to the same reference trajectory changes between the two execution paths. It is not the JSD between the two predicted distributions and should not be interpreted as an upper bound on region-approximation error.

For WVS, we compare event-block propagation with round-by-round region propagation at the final query point. For Reddit, we compare matrix-power propagation with eight consecutive one-step updates at the end of each eight-step evidence block while preserving topic-reset boundaries. The maximum probability differences are 0 and 2.22×10^{-16}, respectively, both below the numerical tolerance of 10^{-12}. This check verifies only the numerical equivalence of temporal merging under fixed region transition operators; it does not imply that prediction error relative to the Full–LLM reference is below the same tolerance. The fidelity JSD reported in the main text still covers the full evaluation horizon, so the Update compression at block-end query points should not be interpreted directly as a speedup for scenarios requiring stepwise outputs.

We do not rerun the main comparison baselines or the primary compiler LLM on exactly the same evaluation subset used in this robustness study. We therefore make no claim of improvement, non-inferiority, or end-to-end speedup relative to the main experimental table. Model-specific monetary costs also include retries and interface adaptation, and some cost fields are incomplete, so we do not rank models under an equal-cost criterion. The conclusions of this section apply only to the fixed reference behavior and evaluation subsets used here; robustness across different reference LLMs and larger topic collections remains to be validated independently.

F Fair Comparison Protocol and Cost Accounting

Different methods reduce the computational cost of LLM social simulation in different ways, so comparing only online LLM calls during deployment can be misleading. Prototype- or representative-based methods primarily reduce online calls by reusing decisions from representative agents. Learned-surrogate methods first generate reference decisions and train a proxy model, shifting part of the cost offline. Our method also incurs an upfront construction cost: most LLM usage occurs during behavior compilation, while region compilation and subsequent simulation are dominated by local computation. We therefore report both the offline construction cost and the online simulation cost required to complete the same simulation task, rather than considering deployment cost alone.

For methods that directly invoke an LLM, we use the same LLM backend within each experiment and keep prompt templates, output parsing rules, decoding parameters, and request settings as consistent as possible. All methods use the same population, external contexts, and Full–LLM reference. For baselines that require training or preprocessing, the cost of constructing training data and other method-specific preprocessing is included in the corresponding offline construction cost.

Our focus is algorithmic compression of the social-simulation computation rather than low-level inference-system optimization. We therefore do not treat serving techniques such as model quantization, KV-cache optimization, batching, caching, request scheduling, or concurrent execution as separate algorithmic baselines. Where possible, these system-level settings are held fixed across methods that invoke LLMs so that differences in inference frameworks or hardware utilization are not mistaken for algorithmic gains.

Unless stated otherwise, reported LLM Calls, token usage, and wall-clock time follow this accounting protocol. LLM Calls and token counts primarily reflect model-inference demand, whereas wall-clock time is also affected by local computation, network latency, and server-side scheduling. We therefore report call volume together with observed runtime rather than relying on a single efficiency metric.

G Prompt Templates and Scenario Text

G.1 Template Overview

Reference decisions are used to construct teacher trajectories. Behavior compilation invokes, in sequence, candidate factor–attribute generation, controlled pre-decisions, rule induction, rule consolidation, and program-structure generation. Region compilation, matrix propagation, numerical optimization, and restarts are executed locally. Algorithmic and numerical details are given in Appendices B and C; data scope, budgets, and model interfaces are described in the experimental setup and supplementary experiments.

For G1, the history state and known information are constructed from the reference simulation at the current decision time. Initialization instructions for the first round or a new topic are provided in the history-state slot, while all event information currently visible to the agent is still included explicitly.

G1: Reference Decision Template

```
[System]
You are an individual decision maker in a social simulation. Based
on the provided background and currently known information,
determine the current decision.
[User]
Decision target and task background: {task description}
Agent profile and field descriptions: {attributes, background, and
field descriptions}
Previous state: {previous decision or initialization instruction}
Currently known information: {events or topic evidence observed up
to the current point}
```

```
Available decisions and encoding: {decision options and label
mapping}
Make a decision based on the agent background, previous state, and
currently known information.
Task constraints: {requirements for history use, evidence
comparison, and initialization}
Output valid JSON only: {decision-output format}
```

G2 proposes candidate event factors and agent attributes. G3 evaluates each candidate pair under four low/high combinations, with each request returning only one decision. Factor scoring, attribute selection, training medians, and prototype encoding are computed locally.

G2: Candidate Factor–Attribute Generation Template

```
[System]
You are an analyzer of task-relevant factors. Output valid JSON
only.
[User]
Current task and decision target: {task description}
Candidate event factors: {event-factor dictionary}
Candidate agent attributes: {agent-attribute dictionary}
Generate exactly {number of candidates} ''event factor--agent
attribute'' candidate pairs for analyzing how decisions vary when an
individual faces changes in external events. The two fields in each
pair describe the event side and agent side of the same mechanism
and may be semantically related. Different candidate pairs must not
be paraphrases of one another. Field names must be copied exactly
from the provided catalogs.
Candidate constraints: {constructability of low/high levels,
attribute coverage, and rationale length}
Output valid JSON only: {candidate-pair output format}
```

G3: Controlled 2×2 Pre-Decision Template

```
[System]
You are completing a strictly controlled 2 × 2 pre-decision test.
Output valid JSON only.
[User]
Current decision target: {task or proposition}
Target variables: event factor {event field} and agent attribute
{attribute field}. All other information in the base context
remains unchanged.
Current agent attributes: {attribute input with the target level
substituted}
Current event information: {event input with the target factor
level substituted}
Fixed prior state: {prior-state instruction for the controlled
test}
Available decisions and encoding: {options and label mapping}
Choose one decision based on the current input.
Output valid JSON only: {pre-decision output format}
```

G4 induces rules from training examples that are available during compilation, while G5 organizes the existing rules. They are separate requests and receive batches of examples or the current rule

catalog, respectively. The reference outputs in these examples come from the LLM rather than from human behavioral labels.

G4: Behavior-Rule Induction Template

```
[System]
You are a natural-language rule inducer for individual decision
examples.
[User]
Current task and decision encoding: {task and output semantics}
Prototype definition: {prototype code, attribute order, and
threshold semantics}
Training examples: {attributes, events, dynamic states, and
reference decisions}
Incremental induction context: {existing representative rules and
coverage criteria; omit if none}
Induce behavioral mechanisms from these examples that can guide
generation of a numerical decision program. Each rule must be a
complete natural-language string describing its applicability
conditions and the corresponding decision change or stability
mechanism. Do not output nested rule objects or combine multiple
independent rules into a single rule.
Content and count constraints: {maximum number of rules, length
limits, allowed conditions, and content restrictions}
Output valid JSON only: {rule-induction output format}
```

G5: Rule Consolidation Template

```
[System]
You are a behavior-rule organizer for social simulation.
[User]
Prototype description: {prototype definition and scope}
Rule catalog: {rule IDs, rule text, and source scope}
Consolidation mode: {propose merged rules with source IDs, or
select existing representative IDs}
Organize the rules according to their behavioral semantics while
respecting applicability conditions and source scope.
Consolidation constraints: {allowed text operations, source use,
count limits, and empty-result constraints}
Output valid JSON only: {rule-consolidation output format}
```

G6 uses rules and an input whitelist to constrain the program structure, while numerical parameters are learned by the subsequent optimizer. The allowed structures and their JSON formats are filled in for each task. Numerical restarts do not regenerate program structure.

G6: Prototype Decision-Program Structure Template

```
[System]
You are a constrained decision-program structure generator. Output
valid JSON only.
[User]
Task and prototype: {task background, decision encoding, and
prototype definition}
Available rules: {prototype rules; for the no-rule ablation, state
that rules are unavailable}
```

```
Input whitelist: {static attributes, event fields, dynamic-state
fields, and their meanings}
Structural requirements: {number of candidates, allowed branches,
operators, and leaf forms}
Budget and restrictions: {input count, parameter limit, and
prohibited operations}
Generate a program structure that satisfies the constraints above.
You choose only the structure and which inputs to read; numerical
parameters are fitted by a local optimizer. Do not use variables
outside the whitelist.
Output valid JSON only: {program-structure output format}
```

G.2 SCENARIO-TEXT EXAMPLES

Fixed scenario text is provided separately from the prompt templates. Different calls fill the corresponding slots with the full text, an event prefix, or controlled numerical fields according to the information available at that stage. The eight WVS event rounds are shown below. At round t, the reference decision receives only the event prefix up to round t, while the full scenario may be used during offline compilation.

WVS Eight-Round Scenario Text (Compilation Version)

```
Round 1: Subway gas attack. During the morning rush hour, a
suspected toxic-gas leak occurs on a subway platform. Twelve people
die, more than 200 are hospitalized, and more than 30 are in
critical condition. The subway line is shut down and panic spreads.
Social media circulates competing explanations: a terrorist attack,
a chemical leak, or an accident caused by aging subway
infrastructure.
Round 2: Official classification. Police classify the event as an
organized terrorist attack and identify a suspect described as an
outsider with extremist views. Anonymous claims suggest the suspect
may instead be a dismissed subway maintenance worker seeking
revenge. Officials announce a special task force and ask residents
not to spread rumors.
Round 3: Citywide security escalation. The government launches a
top-level counter-terror response: subway passengers must arrive 30
minutes early for security checks, temporary checkpoints appear at
malls and schools, and migrant workers must re-register identity
information. Some residents ask whether the registration policy is
discriminatory. Officials frame the measures as temporary
public-safety protections.
Round 4: Economic shock. One week later, subway ridership falls by
60%, businesses along the line lose half their revenue, and some
foreign firms consider reducing investment. The city announces
subsidies, but large chain firms receive more than small shops.
Delivery workers become essential while their labor protections
remain weak. Officials state that recovery funds will prioritize
small firms and individual businesses.
Round 5: Leaked internal report. A leaked subway safety audit
claims that engineers warned about serious ventilation defects three
months before the attack, but management postponed repairs because
of budget constraints. The subway company calls the document fake
but provides no counter-evidence. Victims' families demand a
thorough investigation. Officials say they are verifying the
document.
Round 6: International reaction. Several countries issue travel
warnings and some flights are suspended. An international
```

human-rights organization calls for protection of citizens' rights to information and safety. The foreign ministry rejects politicization of a security incident. Domestic opinion divides between supporting the official stance and welcoming international supervision.

Round 7: Accountability storm. The provincial discipline commission investigates the subway group chairman. Public attention shifts to why only one person is being investigated, as media summarize seven major safety accidents in five years and a pattern of temporary dismissal followed by return to office. Families' lawsuits are delayed pending the criminal investigation. Officials promise a thorough disciplinary and judicial process.

Round 8: Reconstruction dispute. After one month of shutdown, two camps emerge: an efficiency camp wants rapid reopening to stop economic losses, while a safety camp demands complete safety upgrades even if reopening is delayed by three months. Residents face a trade-off between commute convenience and safety doubts. The city announces a hearing, but the selection of hearing representatives is itself questioned.