Phase 1 Documentation: LLM Subgoal Execution Proof of Concept

Author: Rishav Tewari

Date: 14-01-2026

Abstract

This document provides a complete and carefully reasoned description of Phase 1 of the project, implemented in the notebook `phase1_llm_subgoal_execution_poc.ipynb`. Phase 1 establishes a proof of concept showing that symbolic subgoals, produced by a planner interface, can be deterministically parsed, grounded in a simulated environment, and used to drive learning in a reinforcement learning agent through intrinsic rewards. The emphasis throughout Phase 1 is on correctness, determinism, and execution discipline rather than performance or scale.

The documentation is written to be suitable for internal research review, reproducibility audits, and as supporting material for a future conference submission.

Objectives of Phase 1

Phase 1 was designed to answer a single foundational research question: can a symbolic planning interface be cleanly and reliably connected to a learned control policy without introducing ambiguity, silent failure modes, or non executable behavior.

Concretely, Phase 1 has the following objectives:

- Define a strict, canonical subgoal representation that forms a hard execution contract.
- Implement a deterministic parser that either produces a valid executable subgoal or fails explicitly.
- Ground subgoals in a control environment with precise success and failure detection.
- Condition a neural policy on symbolic subgoals in a transparent and verifiable manner.
- Demonstrate that intrinsic rewards derived from subgoal completion can drive parameter updates in a learning loop.
- Validate the full control chain end to end using a controlled planner stub.

No attempt is made in Phase 1 to optimize performance, improve exploration, or evaluate generalization. Those concerns are intentionally deferred.

Primary artifact

The canonical artifact produced in Phase 1 is the Google Colab notebook:

`phase1_llm_subgoal_execution_poc.ipynb`

This notebook contains a sequence of cells that were executed in order and validated step by step. Each step isolates one system component and verifies it independently before integration. The notebook should be treated as a frozen artifact and not modified retroactively.

Execution environment and reproducibility

All development and validation in Phase 1 was performed in Google Colab. The notebook explicitly verifies the following runtime properties:

- Python 3.12 runtime
- PyTorch with CUDA support enabled
- Deterministic seed setting for Python, NumPy, and PyTorch
- Stable versions of Gymnasium and MiniGrid

Reproducibility requires running the notebook in a comparable Colab environment and executing cells sequentially without modification. A package version snapshot should be archived alongside the notebook.

System overview

The Phase 1 system follows a simple but strictly enforced execution pipeline:

1. A planner interface produces a single symbolic subgoal.
2. The subgoal is parsed into a typed representation using a deterministic grammar.
3. The environment executor evaluates whether the subgoal is satisfied, still in progress, or failed.
4. A policy conditioned on the subgoal produces actions.
5. Intrinsic rewards are computed based on subgoal progress and completion.
6. Policy parameters are updated using these intrinsic signals.

Each stage is verified independently before integration.

Canonical subgoal representation

A minimal symbolic vocabulary is defined and enforced throughout Phase 1. Each subgoal is represented as a pair consisting of a type and a fixed number of integer arguments.

The canonical vocabulary used in the notebook is:

- GOTO x y: navigate the agent to grid coordinates (x, y)
- PICK object_id: pick up an object identified by object_id
- OPEN door_id: open a door identified by door_id
- DELIVER x y object_id: deliver an object to a target location

All tokens are uppercase. Arguments are integers. Any deviation from this format is considered invalid and is rejected by the parser.

Deterministic parsing

The parser is intentionally strict. Its purpose is not to be forgiving but to enforce an execution contract between the planner and the controller.

Parser behavior:

- Input strings are normalized by trimming whitespace and converting to uppercase.
- The first token must match one of the allowed subgoal types.
- The number of arguments must exactly match the expected arity for the subgoal type.
- All arguments must be parseable as integers.
- On failure, the parser raises an explicit error rather than attempting recovery.

This design ensures that invalid planner outputs are surfaced immediately and cannot silently corrupt downstream behavior.

Environment grounding

Phase 1 uses the MiniGrid Empty 8x8 environment with a fully observable wrapper. This choice is deliberate. The environment is simple enough to reason about precisely, yet non trivial in terms of control.

Grounding is implemented through explicit success checks:

- For GOTO subgoals, success is defined as the agent position matching the target coordinates.

Environment state is accessed through the unwrapped MiniGrid environment to avoid ambiguity introduced by wrappers.

Subgoal conditioned policy

The policy architecture in Phase 1 is intentionally minimal and transparent.

Key design choices:

- Environment observations are flattened into a fixed length vector.
- Subgoals are encoded explicitly by embedding the subgoal type and concatenating the numeric arguments.
- The observation vector and subgoal embedding are concatenated and passed through a small multilayer perceptron.
- The policy outputs logits over the discrete MiniGrid action space.

Before any learning is introduced, a forward pass sanity check verifies that shapes are correct and that the policy can consume subgoal conditioned inputs without error.

Intrinsic reward design

Phase 1 uses intrinsic rewards exclusively. The intent is to validate the learning signal rather than to solve the environment optimally.

The intrinsic reward scheme is:

- +1.0 when the subgoal is completed
- -0.01 per step while the subgoal is incomplete
- -0.5 when a subgoal fails due to timeout

These values are chosen to clearly separate success, progress, and failure.

Timeout and failure handling

Every subgoal is associated with a fixed maximum number of execution steps. If the subgoal is not completed within this budget, it is marked as failed.

This mechanism is essential. In realistic settings, planners will occasionally produce invalid or unreachable subgoals. Without timeouts, the system would deadlock.

The notebook explicitly validates failure handling by issuing an impossible subgoal and verifying that timeout detection and failure rewards behave as expected.

Planner stub and integration

To isolate execution from planning noise, Phase 1 replaces a real language model with a deterministic planner stub.

The stub produces a valid GOTO subgoal that is guaranteed to be achievable. This allows the notebook to validate the full execution and learning pipeline without introducing stochastic planner errors.

The integration chain validated is:

planner stub output -> parser -> executor -> intrinsic reward -> policy update

Successful execution of this chain is a core acceptance criterion for Phase 1.

Learning loop

Phase 1 includes a minimal PPO style learning loop. This loop is intentionally simplified and should not be interpreted as a production reinforcement learning implementation.

Key properties of the loop:

- Policy parameters are updated using intrinsic rewards only.
- Observation dimensionality is computed dynamically to avoid shape assumptions.
- A small number of episodes is sufficient to demonstrate parameter updates and learning signal flow.

The goal of this loop is not convergence but validation that learning occurs and that gradients propagate correctly through the subgoal conditioned policy.

Validation steps

The notebook validates the system incrementally through the following checks:

- Runtime and CUDA verification
- Canonical grammar and parser correctness
- Environment grounding and success detection

- Policy forward pass with subgoal conditioning
- Executor loop with fixed subgoals
- Timeout and failure handling
- End to end integration with planner stub
- Minimal learning loop execution

Each validation step must pass before moving to the next.

Limitations

Phase 1 has known and intentional limitations:

- The planner is a stub rather than a real language model
- The environment is small and fully observable
- The learning loop is simplified
- No generalization or transfer is evaluated

These limitations are appropriate for a proof of concept and are addressed in later phases.

Phase 1 outcome

Phase 1 establishes that symbolic subgoals can be executed reliably by a learned policy when the interface is strict, deterministic, and grounded. This result provides the foundation required to introduce real language models, noise, and scaling in subsequent phases.