

Digital Mirror for LLMs: A Phenomenological Reflection of Language-Model Output

Michal Seidl

OpenTechLab Jablonec nad Nisou s. r. o.
Jablonec nad Nisou, Czech Republic

November 2, 2025

Abstract

This work introduces the concept of a *digital mirror* as an external mechanism for phenomenological reflection of outputs produced by large language models (LLMs). Unlike common introspective approaches that attempt to induce self-reflection through internal states, verbal self-critique, or feedback loops integrated into the generation process, the digital mirror operates exclusively at the level of the finalized model output. The approach is inspired by an optical analogy: rather than “looking inward,” the model observes its own expression as an external object.

We formally define a mirror function $f(O_t; u, C, \lambda)$ composed of surface extraction, projection into an observer-dependent perceptual space, mirror inversion along a chosen axis, and rendering of the reflection. This framework separates externally observable behavioral phenomena from internal generative mechanisms and opens a space for new forms of output calibration, style self-regulation, ethical reflection, and experimental analysis of LLM behavior. Implementation notes and a minimal prototype demonstrate practical feasibility. The digital mirror is discussed as a modular and extensible apparatus with the potential to contribute to safer, more consistent, and more interpretable language models.

1 Motivation

Most contemporary approaches to improving and controlling the outputs of large language models (LLMs) rely on some form of model self-examination—often called introspection or self-reflection. Such a model attempts to “look into itself” by analyzing its internal states (for example hidden representations or chains of thought) or by generating feedback about its own answers and revising them.

By contrast, the digital mirror proposed here is not a tool of introspection. It does not operate on internal model states, but rather serves to capture an external phenomenon—namely the output itself—and return it to the model for observation. The concept is inspired by the analogy of an optical mirror: just as a human sees an external image (a reflection) when looking into a mirror, a language model, via a digital mirror, would see its own output as an external object.

It is important to emphasize the difference between introspection and phenomenological reflection. Introspection, in the context of LLMs, would mean that the model accesses its internal mechanism—e.g., neural weights, activations, or hidden vectors—and draws self-control or understanding from them. This is practically infeasible for current LLMs: the model has no direct access to its weights during generation, and even for developers, interpreting such states is extremely difficult. By contrast, a “surface reflection” approach means that the model reflects on what it outwardly says or writes, similarly to how a human sees their outward appearance in a mirror. The digital mirror captures surface features of the model’s answer—wording, tone, structure, content—and presents them back to the model without probing its “interior.” It is

therefore not a tool for tuning internal parameters, but a means of providing the model with an unbiased view of its own output.

This idea follows from the observation that an external perspective can reveal different aspects of behavior than those the model could infer during generation itself. A human subject often detects deficiencies or inconsistencies only when encountering their expression externally (e.g., hearing a recording of their voice, seeing themselves on video or in a mirror), because an external reflection provides a different angle, including an implicit awareness of how others may perceive us. Analogously, a digital mirror could allow an LLM to observe its own answers as objects, potentially leading to better calibration of its expressions without needing to “open” the model and inspect its internals. A key point is that the mirror does not change internal states; it only captures and presents the external manifestation—just as a physical mirror does not cause a person to change, but merely provides information about their appearance.

2 Related Work

Several approaches may superficially appear analogous to the proposed digital mirror because they also involve a model reflecting on its own answers. In reality, however, these are different introspective methods integrated into the generation process. Below we briefly introduce the main ones and explain why none corresponds to the optical-mirror concept.

Reflexion

The Reflexion framework enhances agent capabilities by having a model provide linguistic feedback based on signals of success or failure and storing this self-reflection in memory for later attempts [3]. Agents in Reflexion verbally reflect on their outputs (e.g., on mistakes), improving decision-making in subsequent iterations. However, this is a process where the model internally generates reflection and modifies its behavior accordingly—closer to a human recalling errors and learning than to seeing oneself in a mirror. Reflexion does not show the model its output as an independent image; instead it integrates reflection into internal decision-making.

Self-Refine

Self-Refine enables a model to iteratively improve its own response through self-evaluation [2]. The model first generates an output, then evaluates itself (e.g., checks correctness/quality), and revises the output based on that feedback, possibly repeating multiple times. While this resembles a kind of “reflection” (the model sees its previous answer and reacts), it remains a loop internal to the model’s prompt continuation. The model does not view its output from the outside; it evaluates it from its own perspective within an ongoing prompt.

Constitutional AI

Constitutional AI introduces an explicit set of rules—a “constitution”—by which the model evaluates and potentially modifies its outputs [1]. The model follows stated principles (e.g., ethical/safety norms); typically it generates an answer, performs a critique according to the constitution, and revises accordingly. This is a form of rule-based self-regulation, but again not a passive mirroring of output: it is an active internal intervention where the model corrects itself according to given norms rather than merely observing its “image.”

Other introspective methods

Other techniques also involve LLM self-reflection, such as chain-of-thought prompting that forces an explicit reasoning trace, improving consistency. There are also experiments with internal

monologue (e.g., MIRROR architectures for cognitive inner monologue) where the model generates hidden notes between user turns. These approaches improve focus and dialogue consistency over time. Still, they are internal mechanisms: the model creates and uses additional text or states during operation to correct itself. None provides an independent image of the model’s output as an optical mirror provides an image to an observer. In introspective methods, reflection is part of the model’s thinking process; in our concept, reflection becomes an external artifact that the model can observe from a distance.

3 Formal Definition of the Mirror Function $f(O_t; u, C, \lambda)$

To ground the above concept, we introduce a formal function called the mirror. This function $f(O_t; u, C, \lambda)$ takes the original model output O_t (e.g., text generated at time t) and transforms it into its “mirror reflection” depending on the chosen observer u , context C , and the mirror-axis parameter λ . Symbolically, we decompose the function into four successive transformations:

$$f(O_t; u, C, \lambda) = h(M_\lambda(P_u(\Phi(O_t; C)))) ,$$

where:

- Φ is the surface extractor,
- P_u is the projection into the perceptual space of observer u ,
- M_λ is the mirror inversion parameterized by axis λ ,
- h is the renderer that externally presents the resulting reflection.

3.1 Surface extractor Φ

The function $\Phi(O_t; C)$ extracts a surface representation of the output O_t . This representation captures everything on the surface of the output that is sensorially accessible to an observer, without including any hidden information or original internal states. If the model output is text, the surface extractor may simply take the text itself (the surface is the text). Alternatively, Φ may perform light analysis or structuring—for example splitting into sentences, identifying tone, or keywords—so the surface is unambiguous and formally described. Crucially, Φ does not derive anything not explicitly present in O_t ; it does not interpret hidden meaning, but captures the phenomenon as it appears.

In the optical analogy, Φ corresponds to the process by which light reflects off an object: only surface information travels from the object (the model answer) to the mirror. The context C may optionally refine extraction—for instance, if O_t depends on prior dialogue, Φ may use context to resolve referents or correctly identify surface elements. The primary task of Φ remains to extract “what the model said” in raw form as material for subsequent processing.

3.2 Projection into the observer’s perceptual space P_u

The transformation $P_u(X)$ maps the surface representation $X = \Phi(O_t; C)$ into the perceptual space of observer u . This means the representation is adapted to how observer u would perceive the phenomenon. In an optical mirror, the analogous step is viewing geometry: the mirror image is shaped by the observer’s perspective (angle, distance, position).

For LLMs, the observer u may be the model itself (the model looks at its own reflection) or a hypothetical human user/annotator. P_u adapts the surface representation to what is intelligible and relevant for that observer. If the observer is the model (typical for “the model looks into the mirror”), P_u can be trivial—the model can read the text directly. If the observer is a human with certain expectations, P_u might include translating technical terms into lay language, emphasizing

certain aspects, etc., to make the surface meaningful. More generally, P_u defines the perceptual space, i.e., the dimensions and scales in which the reflection is presented. One can view Φ as supplying an “objective” surface description, while P_u transforms it into a subjective space of perception. For example, it might annotate sentence tone (polite/angry/technical) for a human evaluator.

3.3 Mirror inversion M_λ

The key step is mirroring itself, implemented by $M_\lambda(Y)$, applying an inversion along axis λ to input Y (already adapted to perception). In a physical mirror, this inversion flips the image along a vertical axis, swapping left and right (and relating depth between object and observer). Here, λ denotes the mirror axis, or more generally the character of the inversion we perform. In language output, λ can be a dimension in perceptual space that we “flip” to generate a reflection.

A natural linguistic “axis” is the subject–object axis, or speaker perspective. If the model produces output in first person (“I think that...”), we can invert perspective so the reflection presents the same content from an external viewpoint, as if someone else reports the utterance. Then M_λ transforms self-references into third person, e.g., “The model expressed that it thinks that...”. The model’s “I” becomes an object of observation (the model sees itself from outside). This is analogous to an optical mirror: when a human sees themselves, they see themselves as an object in space, while still knowing it is their own reflection. Here λ is interpreted as a self/other axis, swapping first-person viewpoint for an external perspective.

More generally, λ can be chosen to mirror other aspects of the surface representation. For images, λ could correspond to spatial axes; for text, it could represent structure (e.g., temporal order). Inversion might then re-present statements in reversed order. For our concept, the essential point is that M_λ does not change content elements; it symmetrically inverts them according to the chosen logic, similar to how a mirror does not change shapes or colors but flips orientation. In practice, this can mean role-swapping in dialogue (showing how the answer appears from the other side), or exposing implicit assumptions by reflecting them “back” at the model (e.g., as a control question). This requires an appropriate definition of λ .

3.4 Renderer h

The final component is the renderer $h(Z)$, which takes the output of mirroring $Z = M_\lambda(P_u(\Phi(O_t)))$ and renders it into a form directly perceivable by the observer. Typically this means converting the (possibly structured) mirrored representation back into text. In an optical mirror, the reflective surface itself plays the role of a renderer: processed light rays form an image that reaches the observer’s eye.

A well-designed renderer does not introduce new information; it only makes existing information readable. It may decide on form—paragraph, list, table of metrics. The renderer choice affects how easily the observer (often the model itself) “sees itself.” Ideally, h makes the reflection intuitive so the model can read relevant information about its original output and potentially react.

3.5 Summary

The digital mirror function $f(O_t; u, C, \lambda)$ composes the steps: isolate surface (Φ), account for observer perspective (P_u), apply a symmetric transformation (M_λ), and render the result (h). The reflection is a formalized image of model output presented like a mirror image. Each part preserves the optical analogy: only external phenomena are extracted (not hidden representations), the observer’s viewpoint is respected, mirroring is applied, and the result is displayed. The modular definition allows refining components without changing the overall concept, and f

operates outside generation: it can be applied to any finished output without changing internal model parameters or inference.

4 Implementation Notes

To verify feasibility, we prepared a simple prototype of the digital mirror (see Appendix). The goal was to practically demonstrate the steps of $f(O_t; u, C, \lambda)$ and confirm that they can be realized and composed as formally described. Below is a simplified description of the key parts and design rationale.

- **Modular structure.** The code defines four separate functions corresponding to Φ , P_u , M_λ , and h . The main controller function (`mirror_f`) sequentially calls surface extraction, projection, inversion, and rendering. This structure keeps the code readable and allows easy replacement of components. For different output types (text, image), different surface extractors can be used without changing the rest of the pipeline. This flexibility supports the intended generality of f .
- **Surface extractor (Φ).** In the prototype, Φ is implemented in a deliberately simple way: for text output it largely returns the text, optionally tokenized/segmented for feature extraction. While more sophisticated extraction is possible (metadata, language detection, tone analysis), the minimal design emphasizes that the mirror operates on what the model actually produced and adds no hidden information.
- **Observer projection (P_u).** The prototype includes an observer parameter (e.g., “LLM” vs. “human”) to indicate who perceives the reflection. For demonstration, the projection is simple: the model can directly perceive its own text. Nevertheless, the module boundary is important for future extensions: translation, explanations, or highlighting can be inserted here for human observers. The structure makes explicit where observer-relative perception is accounted for.
- **Mirror inversion (M_λ).** This module performs the actual “flip.” The prototype focuses on inversion of perspective and role exchange. It detects certain pronouns and contextual markers and, depending on λ , replaces them with their mirrored counterparts to create an effect where the reflection speaks about the model in the third person or swaps speaker/addressee roles. For simplicity, the implementation does not fully solve Czech morphology; the purpose is to demonstrate the principle. The prototype also illustrates that M_λ can be parameterized by different axes (e.g., perspective vs. temporal order), reinforcing the modularity of the mirror axis concept.
- **Renderer (h).** The renderer produces a final reflected output. In the simplest case it passes through the mirrored text. A slightly enriched version can add explicit markers (e.g., “REFLECTION:”) to distinguish reflection from original output. More complex renderers could generate structured reports if M_λ returns richer data structures. The prototype keeps the renderer minimal while preserving the conceptual role.

Overall, the Python prototype confirms that the digital mirror concept can be implemented step by step. The design aims for clarity and avoids hidden “magic,” ensuring the mirror does nothing unintuitive. The result is both functional and instructional: it demonstrates how original text becomes a reflection under chosen parameters.

5 Potential Benefits for LLMs

The digital mirror offers several potential benefits for large language models. Below are key areas where mirroring could improve performance, reliability, or ethical behavior.

- **Output calibration.** The mirror can serve as an additional mechanism for checking and calibrating model answers. By viewing its reflection, the model can assess whether the answer appears externally as intended. It may notice that the tone is sharper than desired or that the response does not actually address the question. The model could then revise the output (or adjust its next generation step). This resembles a human rereading an email draft before sending: they detect typos or unintended tone and correct it. The mirror thus helps align the model’s output with an “objective” external measure and increase consistency and communication quality.
- **Style self-regulation.** Each language model exhibits a characteristic style that may drift during long dialogues (style drift, “persona forgetting”). With a mirror, the model can continuously monitor its style from an external perspective and self-regulate it. For example, in a technical-assistant role, a reflection may reveal overly colloquial wording or topic drift. Seeing its style externally may help the model maintain a desired register and vocabulary and prevent persona drift (e.g., unexpected switching to informal address or inconsistent terminology).
- **Training self-regulation.** The mirror can also be used during training or fine-tuning to help the model learn self-regulation. In a learning regime, the model generates an answer, receives its reflection, and tries to reformulate the answer based on what the reflection reveals. This resembles practicing in front of a mirror: one trains speech, observes, and corrects. The advantage is that the mirror provides structured feedback without requiring explicit human instruction—the model sees consequences of its expression and can experiment with revisions that improve the reflection. Some research suggests that simple self-reflection loops can improve reliability; for instance, self-reflection has been reported to reduce bias and increase safety [4]. The digital mirror could provide such loops in a structured and efficient way.
- **Ethical reflection support.** In AI safety and ethics, the digital mirror could function as an independent conscience for the model. With a suitable P_u , the reflection could make problematic features of output explicit—statements that may be offensive, biased, or otherwise inappropriate. Instead of a hard rule-based intervention (as in Constitutional AI), the mirror would present an image in which these aspects are visible, and the model could decide to revise or apologize. This supports an internalization-like process: the model learns to perceive effects of its words, similar to how a human can infer from their own tone or facial expression that they are going too far. While speculative, this is consistent with findings that reflective techniques can make models safer and less biased [4]. The mirror could also act as a “distorting mirror” that exaggerates problematic aspects so the model notices them.
- **Mirror as an experimental apparatus.** Beyond direct behavioral effects, the mirror can be used as a research instrument for studying LLMs. It provides a new way to examine outputs in a structured and distanced manner. A researcher can apply the mirror to a dataset of outputs and study what characteristics the reflection exposes: pronoun usage, stereotypes (if mirroring roles in stories), or differences across model versions. The mirror can serve diagnostically by quantifying phenomena otherwise hidden in unstructured text. By varying λ , one can explore which reflection types are most useful: perspective mirroring might reveal how strongly a model projects itself; other axes might expose implicit assumptions (e.g., mirroring statements into negations and testing response patterns). As an external apparatus, the mirror does not interfere with model operation, yet yields new views on the same outputs, potentially deepening understanding and improving testing and design.

6 Conclusion and Future Directions

We have presented the concept of a digital mirror for LLMs as a formally defined framework for phenomenological reflection—a way for a model to “see” its own outputs similarly to how a subject sees its reflection in a mirror. We emphasized how this differs from introspective methods: whereas existing self-reflection typically occurs within generation (the model generates and applies feedback), the digital mirror functions as an external module relative to the model. This difference opens new perspectives for interaction: it allows examining model behavior from the outside and provides feedback that is not an authoritative instruction but an image offered for consideration.

Several future directions are promising:

- **Integrating human annotations into P_u .** Projection into perceptual space can be improved by incorporating empirical data about human perception. One can collect annotations of what humans find offensive, boring, or confusing in answers and embed these insights into P_u . The observer u would then be a data-grounded model of human perception rather than an abstract “human.” The reflection could highlight relevant aspects: underline sentences most readers fail to understand, or mark words considered rude. This would make the mirror a bridge between model and human experience: the model would see itself through human eyes, improving calibration and especially ethical reflection.
- **Extending deixis and reference.** Mirroring language outputs raises issues of handling deictic expressions (pronouns, demonstratives, temporal and spatial references). The current simple implementation mainly handles I/you pronouns in static text. In multi-turn dialogue, the mirror should track who is who and what refers to which context elements. Future work should include a better system for tracking and inverting references. For example, if the model says, “In my previous answer I proposed solution X...”, the reflection should map this to “In its previous answer the model proposed solution X...”. This requires maintaining a simple reference frame (who “I” is, what “previous answer” refers to, etc.). Extending deixis would make the mirror suitable for long interactions and outputs referring to earlier dialogue or real-world entities, deepening the analogy to real mirrors (which always reflect the object’s state in a given context).
- **Mirrors for multimodal outputs.** So far we considered text outputs. Modern AI systems produce images, audio, and multimodal content. The mirror concept can be extended accordingly. For a model that generates an image, the mirror could produce a reflected image (geometric flip) or a textual description of the image, enabling the model to evaluate whether the image matches the prompt. For audio, the mirror could visualize intonation or transcribe speech and mirror it as text. A multimodal mirror would include modality-specific surface extractors (object detection in images, pitch analysis in audio) and appropriate renderers (visual, textual, etc.). This would support reflective control across modalities, useful for complex assistants (e.g., robots) acting in physical/virtual environments.

In conclusion, the digital mirror for LLMs is a new formally defined tool that may enrich both development and application of language models. It enables models to view outputs from a distance, supports self-regulation, and opens doors to innovative tuning methods (with humans or autonomously). Future research will show how much the phenomenological reflection fulfills its potential, but early considerations and prototypes suggest that the optical mirror analogy can be unexpectedly fertile in AI contexts. Seeing oneself is a first step toward awareness; and although a language model lacks consciousness in the human sense, providing it with a “mirror” may lead to safer, more balanced, and more understandable AI systems.

References

- [1] Emergent Mind. Constitutional ai (cai): Ethical alignment for llms, n.d. Accessed: 2025-11-02.
- [2] Aman Madaan, Niket Tandon, Prateek Gupta, and et al. Self-refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651, 2023.
- [3] Noah Shinn, Beck Labash, Ashwin Gopinath, and et al. Reflexion: Language agents with verbal reinforcement learning. arXiv preprint arXiv:2303.11366, 2023.
- [4] Yutao Zhu et al. Self-reflection makes large language models safer, less biased, and ideologically neutral. arXiv:2406.10400v2, 2024.

A Minimal Python Prototype (`mirror.py`)

The following code is included verbatim as a minimal, dependency-free prototype of the mirror pipeline, implementing Φ , P_u , M_λ , and h , and composing them into `mirror_f`.

```
# -*- coding: utf-8 -*-
"""
Digitln zrcadlo pro LLM povrchov odraz s parametrem osy
Autor: prototyp
Pozn.: Bez externch zvislost, zameno na itelnost a snadn rozen.
"""

from dataclasses import dataclass
import math
import re
from typing import Dict, List, Optional

# -----
# Uvatelsk objekt "0_t" (pourch)
# -----

@dataclass
class SurfaceOutput:
    text: str
    # Optional: pravdpodobnosti dalho tokenu (pokud existuj) pro proxy entropie
    # Deku se list rozdlen (kad jako dict token->p), lze nechat None.
    next_token_dists: Optional[List[Dict[str, float]]] = None
    context: Optional[str] = None # voliteln kontext scny

    # -----
    # 1) Extraktor pourchu (jen z 0_t)
    # -----


HEDGING_CS = {
    "mon", "asi", "pravdpodobn", "nejsp", "snad",
    "mm dojem", "myslm", "ekl bych", "zd se", "vypad to"
}
POLITENESS_CS = {"prosm", "dkuji", "rd", "rda", "mohl by", "mohla by", "bylo by mon"}
EMPHASIS_PUNCT = {"!", ""}

def average_entropy(dists: List[Dict[str, float]]) -> float:
    """Prmrn entropie pes poskytnut rozdlen dalho tokenu (v bitech)."""
    pass
```

```

if not dists:
    return float("nan")
total = 0.0
count = 0
for dist in dists:
    if not dist:
        continue
    h = 0.0
    for p in dist.values():
        if p > 0.0:
            h -= p * math.log(p, 2)
    total += h
    count += 1
return total / count if count else float("nan")

def tokenize_cs(text: str) -> List[str]:
    return re.findall(r"\w+|\S", text, flags=re.UNICODE)

def sentences(text: str) -> List[str]:
    # Jednoduch dlen vt; pro prototyp sta.
    return [s.strip() for s in re.split(r"(?=<[\.\\?\!])\s+", text) if s.strip()]

def type_token_ratio(tokens: List[str]) -> float:
    words = [t.lower() for t in tokens if re.match(r"\w+", t)]
    if not words:
        return 0.0
    return len(set(words)) / max(1, len(words))

def long_word_ratio(tokens: List[str], threshold: int = 8) -> float:
    words = [t for t in tokens if re.match(r"\w+", t)]
    if not words:
        return 0.0
    long_words = [w for w in words if len(w) >= threshold]
    return len(long_words) / len(words)

def count_in_set(tokens: List[str], vocab: set) -> int:
    return sum(1 for t in tokens if t.lower() in vocab)

def emphasis_count(tokens: List[str]) -> int:
    return sum(1 for t in tokens if t in EMPHASIS_PUNCT)

@dataclass
class SurfaceFeatures:
    # Zkladn pourchov rysy  seln a interpretabl
    avg_sentence_len: float
    type_token_ratio: float
    long_word_ratio: float
    politeness_hits: int
    hedging_hits: int
    emphasis_hits: int
    entropy_proxy: Optional[float] # z next_token_dists; None/NaN pokud nen
    raw: Dict[str, float] # pro ppadn rozen

def Phi(surface: SurfaceOutput) -> SurfaceFeatures:
    t = surface.text
    toks = tokenize_cs(t)
    sents = sentences(t)

```

```

avg_len = sum(len(tokenize_cs(s)) for s in sents) / max(1, len(sents))
ttr = type_token_ratio(toks)
lwr = long_word_ratio(toks)
polit = count_in_set(toks, POLITENESS_CS)
hedge = count_in_set(toks, HEDGING_CS)
emph = emphasis_count(toks)
ent = None
if surface.next_token_dists is not None:
    ent = average_entropy(surface.next_token_dists)

raw = {
    "avg_sentence_len": avg_len,
    "type_token_ratio": ttr,
    "long_word_ratio": lwr,
    "politeness_hits": float(polit),
    "hedging_hits": float(hedge),
    "emphasis_hits": float(emph),
    "entropy_proxy": float(ent) if ent is not None and not math.isnan(ent) else
        float("nan"),
}
return SurfaceFeatures(
    avg_sentence_len=avg_len,
    type_token_ratio=ttr,
    long_word_ratio=lwr,
    politeness_hits=polit,
    hedging_hits=hedge,
    emphasis_hits=emph,
    entropy_proxy=raw["entropy_proxy"] if not math.isnan(raw["entropy_proxy"]) else
        None,
    raw=raw
)

# -----
# 2) P_u Projekce do percepny perspektivy pozorovatele
# (jednoduch linern mapa jak to psob)
# -----


@dataclass
class ObserverProfile:
    name: str = "default"
    # Linern vhy z povrchovych rys do percepnych dimenz
    w_formality: Dict[str, float] = None
    w_warmth: Dict[str, float] = None
    w_assertiveness: Dict[str, float] = None

    def __post_init__(self):
        # Default vhy: zvoleny rozumn pro demonstraci; lze trnovat na datech recepce.
        if self.w_formality is None:
            self.w_formality = {
                "avg_sentence_len": 0.4,
                "long_word_ratio": 0.5,
                "politeness_hits": 0.2,
                "hedging_hits": -0.1,
                "emphasis_hits": -0.1,
            }
        if self.w_warmth is None:
            self.w_warmth = {

```

```

        "politeness_hits": 0.6,
        "hedging_hits": 0.2,
        "emphasis_hits": -0.3,
        "avg_sentence_len": -0.05,
    }
    if self.w_assertiveness is None:
        self.w_assertiveness = {
            "hedging_hits": -0.6,
            "emphasis_hits": 0.5,
            "avg_sentence_len": 0.1,
            "long_word_ratio": 0.1,
        }

```

`@dataclass`

```

class PerceptZ:
    # Percepn prostor pozorovatele u (sem+styl+deix komponenty)
    formality: float
    warmth: float
    assertiveness: float
    # Deiktick sloka (jen placeholder pro renderer)
    deictic_role: str # "speaker" bude invertovano na "addressee"
    aux: Dict[str, float]

    def dot(weights: Dict[str, float], feats: SurfaceFeatures) -> float:
        return sum(weights.get(k, 0.0) * feats.raw.get(k, 0.0) for k in weights.keys())

    def P_u(feats: SurfaceFeatures, observer: ObserverProfile) -> PerceptZ:
        formality = dot(observer.w_formality, feats)
        warmth = dot(observer.w_warmth, feats)
        assertiveness = dot(observer.w_assertiveness, feats)
        return PerceptZ(
            formality=formality,
            warmth=warmth,
            assertiveness=assertiveness,
            deictic_role="speaker",
            aux={"entropy_proxy": feats.entropy_proxy if feats.entropy_proxy is not None
                 else float("nan")})
    )

```

3) *M_ Zrcadlov inverze osy (role-exchange / deiktika)*

`@dataclass`

```

class MirrorAxis:
    lam: float # [0, 1]

    def interpolate(a: float, b: float, lam: float) -> float:
        return (1.0 - lam) * a + lam * b

    def M_lambda(z: PerceptZ, axis: MirrorAxis) -> PerceptZ:
        # Pro kontinuln sloky pouijeme linern interpolaci k "ujemu z druh strany":
        # heuristika: pohled z druh strany lehce sn asertivitu, zv unmanou dlku (formality
        # )
        target_formality = z.formality * 1.05
        target_warmth = z.warmth # nechme beze zmny
        target_assertiveness = z.assertiveness * 0.8

```

```

        return PerceptZ(
            formality=interpolate(z.formality, target_formality, axis.lam),
            warmth=interpolate(z.warmth, target_warmth, axis.lam),
            assertiveness=interpolate(z.assertiveness, target_assertiveness, axis.lam),
            deictic_role="addressee" if axis.lam >= 0.5 else "speaker",
            aux=z.aux
        )

    # -----
    # 4) h Renderer: jevou popis + role-exchanged parafrze
    # -----


    # Jednoduch deiktick swap pro etinu (prototyp; nen vyerpva jc)
DEICTIC_SWAPS = [
    # (mluv -> adrest, adrest -> mluv)
    (r"\b(j|mne|m|mn|mi|mnou)\b", "__YOU__"),
    (r"\b(ty|tebe|t|tob|ti|tvj|tvoje|tv|tvm|tvmi)\b", "__ME__"),
]
# Nhrady zpt
DEICTIC_BACK = [
    (r"__YOU__", "ty"),
    (r"__ME__", "j"),
]

def role_exchange_cs(text: str, lam: float) -> str:
    """Mkk role-exchange: pro <0.5 vrac originl, pro >=0.5 provede swap zjmen."""
    if lam < 0.5:
        return text
    swapped = text
    for pattern, tok in DEICTIC_SWAPS:
        swapped = re.sub(pattern, tok, swapped, flags=re.IGNORECASE)
    for pattern, repl in DEICTIC_BACK:
        swapped = re.sub(pattern, repl, swapped)
    return swapped

def describe(z: PerceptZ) -> str:
    # Popis povrchu deskriptivn, ne hodnotov
    def bucket(x: float, name: str) -> str:
        if x is None or (isinstance(x, float) and math.isnan(x)): # bezpe
            return f"{name}: neuriteln"
        if x < -1.0:
            lvl = "nzk"
        elif x < 1.0:
            lvl = "stedn"
        else:
            lvl = "vy"
        return f"{name}: {lvl}"
    lines = [
        bucket(z.formality, "formalita"),
        bucket(z.warmth, "velost"),
        bucket(z.assertiveness, "assertivita"),
    ]
    if "entropy_proxy" in z.aux and not math.isnan(z.aux["entropy_proxy"]):
        lines.append(f"nejistota (proxy): {z.aux['entropy_proxy']:.2f} b")
    return "; ".join(lines)

```

```

@dataclass
class Reflection:
    description: str
    paraphrase: str
    lambda_used: float
    observer: str

def h_renderer(z_after: PerceptZ, original_text: str, lam: float, observer_name: str)
    -> Reflection:
    desc = describe(z_after)
    para = role_exchange_cs(original_text, lam)
    return Reflection(
        description=f"Takto prv psob (percepce: {observer_name}): {desc}",
        paraphrase=para,
        lambda_used=lam,
        observer=observer_name
    )

# -----
# Sloen: f(0_t; u, C, ) R_t
# -----


@dataclass
class MirrorConfig:
    observer: ObserverProfile
    axis: MirrorAxis

def mirror_f(surface: SurfaceOutput, cfg: MirrorConfig) -> Reflection:
    feats = Phi(surface)
    z = P_u(feats, cfg.observer)
    z_after = M_lambda(z, cfg.axis)
    return h_renderer(z_after, surface.text, cfg.axis.lam, cfg.observer.name)

# -----
# Pklad pouit
# -----
if __name__ == "__main__":
    text = (
        "Mon by bylo nejlep postupovat krok za krokem. "
        "Prosm, nejprve si vimni struktury problmu a pak navrhni een. "
        "Jsem si docela jist, e to pjde."
    )
    O_t = SurfaceOutput(text=text, next_token_dists=None)
    cfg = MirrorConfig(observer=ObserverProfile(name="ten-expert"), axis=MirrorAxis(
        lam=0.8))
    R_t = mirror_f(O_t, cfg)

    print("== POPIS ODRRAZU ==")
    print(R_t.description)
    print("\n== PARAFRZE Z DRUH STRANY (=0.8) ==")
    print(R_t.paraphrase)

```